PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO PARA A ANÁLISE DE IMPACTO EM CÓDIGO FONTE USANDO ONTOLOGIAS E RECUPERAÇÃO DE INFORMAÇÃO

RODRIGO PEROZZO NOLL

Tese apresentada como requisito parcial à obtenção do grau de Doutor, pelo programa de Pós Graduação em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2012

N793m Noll, Rodrigo Perozzo
Um modelo para a análise de impacto em código fonte usando ontologias e recuperação de informação / Rodrigo Perozzo Noll. – Porto Alegre, 2012.
235 p.

Tese (Doutorado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Marcelo Blois Ribeiro.

1. Informática. 2. Ontologia. 3. Recuperação da Informação. I. Ribeiro, Marcelo Blois. II. Título.

CDD 006.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "Um Modelo para a Análise de Impacto em Código Fonte Usando Ontologias e Recuperação de Informação", apresentada por Rodrigo Perozzo Noll, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, Sistemas de Informação, aprovada em 13/08/2012 pela Comissão Examinadora:

Prof. Dr. Ricardo Melo Bastos – PPGCC/PUCRS

Prof. Dr. Marcelo Blois Ribeiro – Orientador — GE - Brasil

Prof. Dr. Rafael Prikladnicki - PPGCC/PUCRS

Profa. Dra. Renata de Matos Galante - UFRGS

Prof. Dr. José Palazzo Moreira de Oliveira – UFRGS

Homologada em 03/10/12, conforme Ata No. 21 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Para minha esposa e filha,
Sandra e Elizabeth,
pelo amor e inspiração.

AGRADECIMENTOS

Primeiramente, eu dedico esta tese a minha esposa e filha, Sandra e Elizabeth, e a meus pais, Carlos e Sônia. Sem vocês, não estaria aqui hoje e não seria a pessoa que sou. Muito obrigado pela compreensão nos inúmeros momentos de ausência e ansiedade. Seu apoio e carinho foram fundamentais para transformar esta tese em realidade.

Eu gostaria de apresentar minha gratidão ao meu orientador, Prof. Marcelo Blois Ribeiro, por todos os conselhos, confiança, apoio, motivação e agradáveis discussões. Seus ensinamentos e referência como pessoa e orientador serão sempre lembrados.

Também sou grato ao meu segundo orientador, Prof. Ricardo Bastos, por seu apoio e disponibilidade para a condução desta tese, bem como sua importante contribuição durante o desenvolvimento deste trabalho.

Gostaria de prestar meus agradecimentos a Profa. Renata Galante por suas valorosas contribuições durante as avaliações intermediárias deste trabalho.

Agradeço aos membros da banca, professores(a) José Palazzo M. de Oliveira, Renata Galante e Rafael Prikladnicki, por aceitarem o convite para avaliar este trabalho.

Agradeço aos meus colegas do grupo ISEG/PUCRS e ao convênio Dell-PUCRS, por viabilizar a bolsa necessária para a realização desta tese.

Aos amigos, pela motivação e companheirismo ao longo dos anos.

Aos professores e funcionários do Programa de Pós-Graduação em Ciência da Computação da PUCRS.

UM MODELO PARA A ANÁLISE DE IMPACTO EM CÓDIGO FONTE USANDO ONTOLOGIAS E RECUPERAÇÃO DA INFORMAÇÃO

RESUMO

Mudanças são inevitáveis durante o ciclo de vida do software. Estas mudanças são resultado de diferentes necessidades, como a evolução do conhecimento sobre os processos de negócio, alterações de ambiente, etc. Nestas circunstâncias, é crucial ter controle sobre o que essas mudanças representam na aplicação. A análise de impacto representa o processo que gera este conhecimento. Essa análise possui um significado abrangente dentro do desenvolvimento de software, incluindo desde a identificação de estruturas no código fonte até o controle das restrições de gestão de projeto. Esta tese apresenta um modelo para analisar o impacto no código fonte de uma aplicação utilizando ontologias, visando melhorar a precisão e revocação de estruturas identificadas se comparadas a técnicas existentes. O uso de ontologias integra uma perspectiva semântica nas técnicas tradicionalmente baseadas na análise sintática do código fonte. Para o desenvolvimento do modelo de análise de impacto, foram definidos dois submodelos: o de rastreabilidade e o de probabilidade. O modelo de rastreabilidade recebe como entrada o código fonte e uma ontologia de domínio e gera como resultado uma ontologia de rastreabilidade populada automaticamente com elos entre conceitos do domínio (classes e propriedades) e estruturas do código (classes, métodos e atributos). Estes elos são populados através de um analisador léxico e semântico que realiza a categorização, normalização (geração de *tokens*, expansão e eliminação) e comparação (*stemming*). Com base na ontologia de rastreabilidade e em um requisito de mudança, o modelo probabilístico classifica cada elo de rastreabilidade utilizando o modelo de Redes de Crenças Bayesianas. Para o cálculo de probabilidade, a classificação dos nodos utilizou o algoritmo PageRank do Google e das arestas, a análise de frequência TFIDF e a dependência conceitual, definida nesta tese. Este modelo de análise de impacto foi implementado como um *plugin* do eclipse e foi avaliado empiricamente através de três experimentos controlados.

Palavras chave: análise de impacto, ontologia, recuperação de informação.

AN IMPACT ANALYSIS MODEL IN SOURCE CODE USING ONTOLOGIES AND INFORMATION RETRIEVAL

ABSTRACT

Changes are inevitable during the product lifecycle. These changes are due different needs, such as the knowledge evolution of business processes, environment or infrastructure changes, etc. Under these circumstances, it is crucial to have the control about the knowledge of what these changes mean to the system. Impact analysis represents the process that creates this knowledge. The impact analysis has a wide meaning for software development, such as the assessment of source code structures or the control of project management constraints. This thesis aims to present a model to analyze the impact in source code using ontologies in order to improve precision and recall of the identified source code structures compared to existing techniques. The use of ontologies integrates a semantic layer under the traditional techniques founded in syntactic analysis. To develop the impact analysis model, it was defined two different sub models: traceability and probability. The traceability model receives as input the source code and domain ontology and generates a traceability ontology populated with links between domain concepts (classes and properties) and source code units (classes, methods and attributes). These links are populated by a lexical and semantic analyzer that categorize, normalize (token generation, expansion and elimination) and compare (stemming) each token. With the traceability ontology and a change request, the probabilistic model classifies each traceability link using Bayesian Belief Networks. To the probability calculus, the node classification used the Google PageRank algorithm and the arrows used frequency analysis TFIDF and conceptual dependency, defined in this thesis. The impact analysis model was implemented as an Eclipse plugin and was empirically evaluated using three controlled experiments.

Keywords: impact analysis, ontology, information retrieval.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

AST – *Abstract Syntax Tree*

CdU – Caso de Uso

DAO – *Data Access Object*

DC – Dependência Conceitual

DTO – *Data Transfer Object*

EC – Engenheiro do Conhecimento

GQM – *Goal, Question, Metric*

IDE – *Integrated Development Environment*

JDT – *Java Development Tools*

LSI – *Latent Semantic Indexing*

OWL – *Ontology Web Language*

PLN – Processamento de Linguagem Natural

RCB – Redes de Crenças Bayesianas

RdM – Requisito de Mudança

RI – Recuperação de Informação

TFIDF – *Term Frequency–Inverse Document Frequency*

UML – *Unified Modeling Language*

SUMÁRIO

1. INTRODUÇÃO

A análise de impacto avalia os diversos riscos associados a uma mudança [Pfl90]. Essa análise possui uma abrangência significativa dentro de um projeto de software, podendo considerar aspectos econômicos, financeiros, de recursos humanos, entre outros. Para o propósito deste trabalho, a análise de impacto se refere a mudanças em software, mais especificamente, a análise dos efeitos de determinada mudança no código fonte de uma aplicação.

Para uma definição clara do termo “análise de impacto”, este trabalho utilizará a apresentada por Bohner e Arnold que considera a análise de impacto como o processo de identificar potenciais consequências de uma mudança ou a estimativa do que precisa ser modificado para realizar uma mudança [Arn96]. Outras definições serão discutidas na Seção 2.1.1, mas por enquanto é suficiente perceber que essa definição diferencia explicitamente consequências e modificações.

Consequências típicas se referem ao impacto em algumas restrições do projeto, tais como custo, tempo e qualidade, enquanto modificações consideram aspectos de desenvolvimento, no qual estruturas de código devem ser alteradas para atender certa demanda. Modificações podem ter consequências positivas ou negativas, dependendo dos resultados de sua execução. Essas consequências se referem a desvios de funções existentes que podem implementar novas necessidades, bem como gerar falhas no sistema.

Mudanças devem ser analisadas em duas perspectivas: (1) do time de desenvolvimento que implementa a mudança e (2) dos cenários de negócio que representam as partes interessadas, clientes e o mercado em geral. O desenvolvimento de software engloba essas perspectivas através da execução de um conjunto de atividades que traduzem necessidades de negócio em código fonte.

Ao longo dos anos, diferentes estratégias e práticas vêm sendo desenvolvidas para superar as dificuldades inerentes à intersecção dessas perspectivas. Estas práticas geralmente formalizam um conhecimento particular do sistema em artefatos de software, tais como código fonte, especificações e diagramas. Esses artefatos representam uma perspectiva do sistema em um período de tempo e tem o objetivo de compartilhar e alinhar esse conhecimento entre as partes interessadas.

Como softwares modelam essencialmente fenômenos do mundo real, é possível adotar modelos e técnicas de desenvolvimento fundamentadas na semântica [Hol07]. O desenvolvimento orientado a ontologias representa a sinergia entre processos e artefatos da Engenharia de Software e ontologias, que especificam explicitamente conceitos do mundo real [W3c12].

Para viabilizar a análise de impacto em código fonte utilizando uma perspectiva semântica, esta tese inclui modelos de rastreabilidade e de probabilidade. O modelo de rastreabilidade estrutura e relaciona conceitos do domínio com o código fonte da aplicação. Para isso, tanto o código fonte quanto o modelo de domínio são representados e relacionados utilizando ontologias. A rastreabilidade utiliza técnicas de processamento de linguagem natural para extrair conhecimento a partir de dados brutos, populando a ontologia de domínio com elos de rastreabilidade. O modelo de probabilidade, por sua vez, identifica a relevância de cada elo de rastreabilidade, analisando as relações de baixo nível extraídas do código fonte e ponderando com a organização semântica e conceitual modelada na ontologia. O resultado é a identificação de estruturas de código relevantes frente a uma solicitação de mudança.

A perspectiva semântica da análise de impacto inclui a especificação de uma base de conhecimento do projeto de software pela manutenção de ontologias do domínio e do código fonte da aplicação. Estas ontologias mantêm o conhecimento tanto do negócio quanto das entidades computacionais derivadas de seus conceitos. Para apoiar essa tese, os experimentos discutidos em [Nol07a, Nol07b] evidenciam empiricamente que a rastreabilidade de modelos UML orientada a conceitos é mais precisa e necessita de menos esforço do que abordagens tradicionais baseadas em requisitos. Com base nessas evidências, esta pesquisa tem como objetivo desenvolver um modelo capaz de avaliar a relevância das entidades no código fonte impactadas por determinada solicitação de mudança, bem como avaliar as vantagens que podem ser obtidas pelo uso de representações formais do conhecimento durante a manutenção de software.

1.1. Motivação

A análise de impacto é uma atividade fundamental durante a evolução de software, pois cerca de 80% do ciclo de vida de um sistema equivale a sua manutenção [Pfl04]. O uso contínuo de qualquer sistema não trivial estimula um fluxo de mudanças. A

adição de novas funcionalidades provavelmente impactará as funcionalidades existentes, sejam elas manutenções corretivas, adaptativas ou de melhoria [Cha00].

Mudanças podem ocorrer por diversas razões, desde adição de novas funcionalidades, correção de falhas, mudança de ambiente (como troca de servidor de aplicação ou banco de dados) ou pedidos de melhoria. Essas mudanças são consequência do conhecimento que os usuários adquiriram sobre o problema resolvido por um software e esperam que o software evolua tanto quanto esse conhecimento [Has02].

Durante o desenvolvimento de software, diferentes necessidades aumentam sua complexidade, desde questões técnicas até de negócios. As mudanças requeridas no software não devem ser negligenciadas ou implementadas incondicionalmente no sistema. A análise de impacto fornece, neste sentido, meios para determinar as características e efeitos de uma proposta de mudança, fornecendo a base para sua aprovação ou não.

Um fator crítico da análise de impacto diz respeito ao encadeamento de mudanças. A modificação em uma unidade do código fonte geralmente impacta estruturas dependentes, podendo introduzir acidentalmente defeitos. Um problema típico é a dificuldade de identificar essas estruturas, principalmente em sistemas com alto acoplamento e baixa coesão. A análise de impacto, que inclui analisar as dependências entre estruturas, pode ser utilizada para definir o escopo e a complexidade de determinada modificação, bem como os riscos associados.

Modelos atuais para análise de impacto, que incluem análise de rastreabilidade e de dependência, geralmente utilizam apenas a estrutura sintática do código para calcular o particionamento do programa e gerar os grafos de chamadas, conforme apresentado na Seção 2.1. Esses modelos apresentam algumas limitações exploradas nas Seções 2.2.5 e 2.3.5 que poderiam ser melhor resolvidas utilizando um modelo semântico para representação do conhecimento.

Neste contexto, a motivação desta tese é a possibilidade de recuperar estruturas de código com uma melhor precisão (*precision*) e revocação (*recall*), medidas estas exploradas na Seção 2.3.4, se comparada a abordagens existentes de análise de impacto. Esta motivação se apoia na premissa de que o contexto semântico da aplicação, e não apenas a análise sintática do código, pode gerar uma análise mais apurada dos impactos resultantes de uma solicitação de mudança. Para tanto, se sugere o uso de

ontologias como formalismo primário para compartilhar e relacionar conhecimento [Gua98]. Esta camada de semântica viabilizaria uma análise mais flexível e adaptativa, conforme apresentado no Capítulo 3.

1.2. Questão de Pesquisa

Conforme Creswell [Cre94], questões de pesquisa estão mais associadas ao paradigma qualitativo de pesquisa, enquanto o paradigma quantitativo faz uso de hipóteses e objetivos. A presente pesquisa possui tanto métodos qualitativos em sua fase exploratória, quanto métodos quantitativos na fase confirmatória, conforme descrito na Seção 1.4. Para apoiar a fase exploratória e nortear a proposta objeto deste trabalho, foi definida a seguinte questão de pesquisa:

"Como melhorar, em relação às abordagens existentes para análise de impacto, a recuperação de estruturas de código fonte relevantes a partir de uma solicitação de mudança?"

Esta questão de pesquisa engloba o principal aspecto explorado nesta pesquisa, que inclui identificar os impactos no código fonte resultantes de determinada solicitação de mudança. Para esta análise, é proposto um modelo de rastreabilidade utilizando ontologias e um modelo de probabilidade que utiliza técnicas de recuperação de informação para identificação de estruturas relevantes, avaliando a relação entre os termos do domínio e estruturas da aplicação.

1.3. Objetivos

Uma vez definida a questão de pesquisa, definiu-se o objetivo geral e os objetivos específicos deste trabalho. O objetivo geral deste trabalho *é desenvolver um modelo para identificação de estruturas do código fonte impactada por solicitações de mudanças, utilizando uma perspectiva semântica*.

Para a consecução do objetivo geral proposto, identificam-se os seguintes objetivos específicos:

1. evoluir o modelo de análise de rastreabilidade utilizando ontologias apresentado em [NOL07a];

2. desenvolver um processo para população da ontologia pela automação do mapeamento de estruturas do código fonte e conceitos do domínio;
3. desenvolver um modelo de análise de probabilidade utilizando modelos e técnicas de recuperação de informação e grafos de chamadas;
4. desenvolver um protótipo funcional como avaliação da viabilidade do modelo;
5. avaliar empiricamente a tese utilizando experimentos.

1.4. Metodologia de Pesquisa

A metodologia compreende a classificação e definição da pesquisa, bem como a forma de abordar determinado problema, que inclui objetivos, procedimentos técnicos e coleta de dados.

Ao se conduzir uma pesquisa, o primeiro critério a se considerar é sua classificação. Conforme [Whi04], pesquisas podem ser classificadas como confirmatórias, que testam relacionamentos pré-especificados, e exploratórias, que definem possíveis relacionamentos em sua forma mais genérica.

O método de pesquisa exploratório tem como objetivo:

> A definição de questões e hipóteses de um estudo subsequente ou a determinação da viabilidade de procedimentos de pesquisa desejados [Yin03].
>
> Proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a constituir hipóteses. Pode-se dizer que estas pesquisas têm como objetivo principal o aprimoramento de ideias ou a descoberta de intuições. Seu planejamento é, portanto, bastante flexível, de modo que possibilite a consideração dos mais variados aspectos relativos ao fato estudado [Gil02].

De acordo com o descrito, este trabalho é caracterizado em sua maioria como exploratório, pois se pretende desenvolver um modelo para aprimorar técnicas convencionais de análise de impacto utilizando ontologias. A análise de impacto é observada sob uma perspectiva diferente da usual, e a questão de pesquisa que guia este trabalho representa uma questão aberta, reforçando o caráter exploratório. Esta tese tem como base a existência de abordagens que evidenciam os benefícios da rastreabilidade apoiada por ontologias [Nol07a], gerando indícios de que a análise de impacto também poderá se beneficiar deste apoio semântico.

O escopo dessa pesquisa inclui um aspecto descritivo, observando a prática e as limitações relacionadas à análise de impacto e, então, propondo melhorias. Neste contexto, as melhorias são aplicadas com o objetivo de aprimorar as técnicas existentes de análise de impacto.

Oates fornece um sumário de diferentes tipos de produtos derivados de uma pesquisa [Oat06]: evidência, metodologia, conceitos e teorias, análise e produto. Dentre as classificações definidas pelo autor, a estratégia de pesquisa utilizada nesta pesquisa é a chamada “projeto e criação”, que possui como foco o desenvolvimento de novos produtos de tecnologia da informação. Foi escolhido o processo de desenvolvimento prototipal para verificar a viabilidade do modelo de análise de impacto proposto.

Apesar da ênfase exploratória, este trabalho também inclui uma fase confirmatória, pois, conforme [Whi04], “estudos confirmatórios são utilizados para auditar a confiabilidade, validade do conteúdo e do construto”. Neste caso, a avaliação é realizada utilizando experimentos para validar empiricamente o modelo de análise de impacto proposto. Estes experimentos isolam algumas variáveis independentes, no caso diferentes técnicas de análise de impacto, para avaliar quantitativamente as possíveis combinações de resultados através das variáveis dependentes de precisão, revocação e esforço. Esta avaliação do modelo se caracteriza como empírica, pois evidencia o impacto nas variáveis dependentes a partir da manipulação das variáveis independentes [Kum05].

1.4.1. Desenho e fases da pesquisa

Para desenvolver o modelo para análise de impacto proposto, planejou-se uma metodologia de pesquisa organizada em duas grandes fases: exploratória e confirmatória. A Figura 1.1 apresenta o desenho de pesquisa que ilustra as fases adotadas pela pesquisa.

A primeira fase caracteriza-se como exploratória e consiste em duas etapas: (1) Identificação do Problema de Pesquisa e (2) Desenvolvimento do Modelo de Análise de Impacto. A segunda fase caracteriza-se como confirmatória e é composta por uma etapa que inclui o desenvolvimento do protótipo e experimentos.

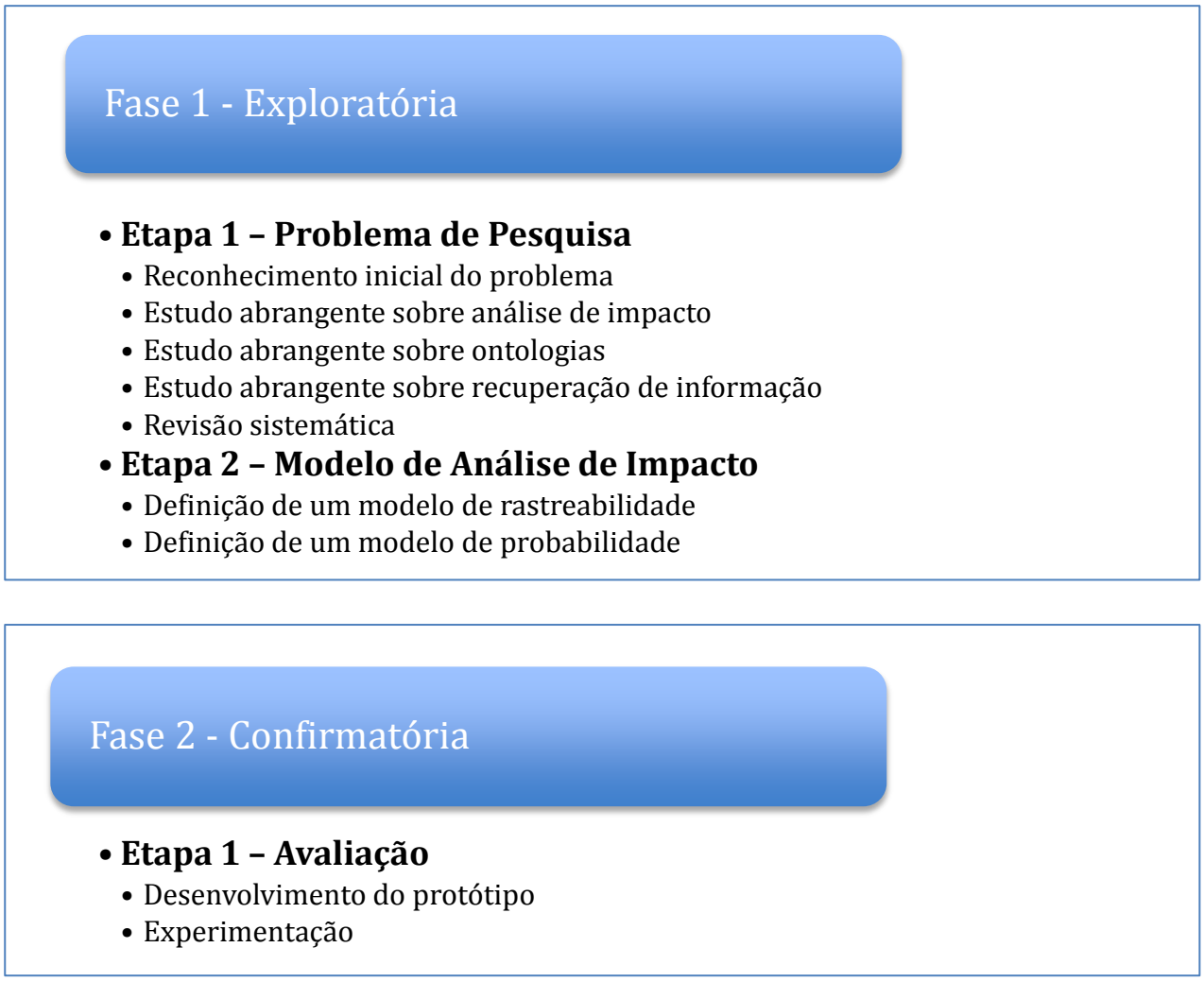


Figura 1.1 – Desenho e fases da pesquisa.

A **Etapa Exploratória 1 – Problema de Pesquisa** foi eminentemente exploratória na qual foi identificado um problema real no contexto de gerenciamento de mudança de software: a análise de impacto. Foi formulado um tópico genérico de pesquisa dentro do problema identificado e executada uma pesquisa em profundidade sobre a análise de impacto, bem como sua relação com ontologias e recuperação de informação. Este estudo teve como objetivo aprofundar o entendimento nos modelos de análise de impacto existentes bem como das propostas que fazem uso de ontologias e recuperação de informação para apoiar a análise de impacto. Após este estudo, foi realizada uma revisão sistemática para identificar a intersecção entre essas duas áreas. O conceito de revisão sistemática foi introduzido na área de Engenharia de Software por [Kit04] e se mostrou bastante eficiente, justificando o seu uso neste trabalho. Com isso, foi possível ampliar a cobertura dos estudos em profundidade e identificar lacunas de pesquisa. O resultado desta etapa foi descrito no Capítulo 2 desse documento.

A **Etapa Exploratória 2 – Modelo de Análise de Impacto** representou o desenvolvimento de um modelo para apoiar a análise de impacto utilizando, para tanto, dois modelos: de rastreabilidade e de probabilidade. Para o modelo de rastreabilidade, as estruturas do código fonte foram relacionadas aos conceitos da ontologia de domínio. Foi definido um modelo de representação de elos de rastreabilidade, flexível o suficiente para suportar quaisquer entidades representadas por artefatos de software. Uma vez definido esse modelo, percebeu-se a necessidade de automação do mapeamento entre código e conceitos, diminuindo os esforços requeridos pela equipe de desenvolvimento nesta tarefa. Foi desenvolvida uma arquitetura utilizando PLN para aprendizagem de ontologia e geração de instâncias de indivíduos a partir do código fonte da aplicação, que representam os elos de rastreabilidade. O modelo de probabilidade pondera as estruturas do código fonte que potencialmente serão impactadas em virtude de determinada mudança. Este modelo considera a dependência entre a organização lógica e de baixo nível da aplicação (código fonte) e as dependências entre os conceitos do domínio. Esse cálculo de relevância foi desenvolvido utilizando um modelo de recuperação de informação de Redes de Crenças Bayesianas que calcula a probabilidade de impacto para cada nodo do modelo de rastreabilidade. Este cálculo inclui a frequência que cada conceito ocorre no código fonte, bem como a dependência conceitual existente entre a ontologia e o código fonte. Essa dependência utiliza o grafo de chamadas em duas vias, isto é, analisa a dependência tanto da ontologia de domínio quanto do código fonte, em oposição às propostas atuais que consideram apenas a dependência do código fonte. Ao final, apresentam-se as classes que potencialmente serão impactadas pela solicitação de mudança;

A **Etapa Confirmatória 1 – Avaliação** teve como objetivo desenvolver um protótipo funcional para avaliar a viabilidade do modelo proposto bem como sua aplicação utilizando experimentos. O processo de experimentação requer um preparo para conduzir e analisar corretamente o objeto de estudo. Este esforço representa seu maior benefício: a capacidade de controlar as variáveis relacionadas às entradas do exercício, permitindo a geração de conclusões significativas para o experimento. Durante a avaliação, foi necessário controlar os indivíduos que estarão aplicando as abordagens de análise de impacto, utilizando tanto o método proposto por este trabalho quanto às abordagens convencionais apoiadas por requisitos. Uma vez realizada a experimentação, foi feita a documentação dos resultados e o empacotamento do experimento, visando possibilitar sua replicação.

1.5. Organização da Tese

Esta tese está organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, que inclui uma visão geral sobre análise de impacto, bem como seu relacionamento com ontologias e recuperação de informação. Este capítulo também apresenta uma revisão sistemática que apresenta o estado da arte relacionado aos objetivos deste trabalho. O Capítulo 3 apresenta o modelo de análise de impacto e seus componentes. Este modelo é exemplificado através de um exemplo prático, ilustrando cada um dos seus aspectos. O Capítulo 4 apresenta a implementação do protótipo desenvolvido para analisar a viabilidade da tese, incluindo sua arquitetura e guia de uso. Este modelo é avaliado empiricamente através de três experimentos controlados conforme apresentado no Capítulo 5. As conclusões, contribuições e limitações da tese são exploradas, por fim, no Capítulo 6.

2. REFERENCIAL CONCEITUAL

Atividades relacionadas à análise de impacto são comuns durante o ciclo de vida de software. Porém a falta de uma definição comum para o termo e o caráter empírico muitas vezes empregado nessa avaliação evidenciam algumas limitações da área [Arn93].

Para o desenvolvimento de uma proposta que analise os impactos de mudanças em uma aplicação, primeiro é necessário entender os conceitos relacionados. A Seção 2.1 apresenta uma revisão sobre o que é análise de impacto no contexto de desenvolvimento de software, bem como alguns modelos de uso. Também são apresentadas as etapas necessárias para essa análise e o contexto associado à evolução de software e gerenciamento de mudanças.

Mudanças em software são geralmente resultado de novas orientações sobre cenários de negócio e conceitos associados. Para modelagem conceitual, diversos pesquisadores vêm sugerindo o uso de ontologias. Com base nessa premissa, a Seção 2.2 relaciona ontologias à análise de impacto, apresentando definições de ontologias e sua integração com o desenvolvimento de software, além de uma descrição sobre linguagens para manipulação de ontologias e seu relacionamento com análise de impacto.

A Seção 2.3 apresenta conceitos sobre Recuperação de Informação (RI). A RI é uma estratégia eficiente para busca de informação não estruturada, como o código fonte da aplicação. Para tanto, serão apresentados alguns modelos e algoritmos que podem ser aplicados à identificação de estruturas de código associadas a requisições de mudanças. Nesta seção também é apresentado o processamento de documentos, algumas métricas para avaliar a eficiência de modelos de RI e trabalhos relacionados à análise de impacto e recuperação de informação.

A Seção 2.4 apresenta uma revisão sistemática visando identificar a intersecção entre as duas áreas foco da fundamentação teórica: ontologia e recuperação de informação no contexto de análise de impacto, visando identificar propostas semelhantes ao objeto desta tese.

Por último, na Seção 2.5, além das considerações sobre o capítulo, é discutido o escopo e abrangência da tese.

2.1. Análise de Impacto

Esta seção discute análise de impacto no contexto de desenvolvimento de software, incluindo sua definição, conceitos e evolução. Também são discutidas as estratégias manuais e automatizadas para analisar impactos, com ênfase nas análise de rastreabilidade e dependência.

2.1.1. Principais Conceitos e Definições

O gerenciamento de mudanças é uma atividade fundamental para a manutenção e evolução de software. Um fator chave desse gerenciamento é determinar quais serão as implicações decorrentes de determinado requisito de mudança [Pfl90]. Essas implicações incluem identificar as partes do sistema que serão impactadas e quanto de código é necessário ser modificado.

Um impacto de uma mudança representa uma parte do software que é determinantemente afetada e que merece investigação. Essa inspeção tem como objetivo manter a estabilidade do software, o que representa a sua resistência a um potencial efeito colateral que um programa poderia ter quando modificado [Yau80]. Esse efeito colateral, por sua vez, é considerado como um erro ou outro comportamento inesperado resultante de uma modificação [Fre81] que geralmente é causado por fazer uma pequena alteração em um sistema que afeta outras partes [Ste79].

Embora a análise de impacto seja praticada há muitos anos, ainda não existe um consenso em sua definição. O Glossário de Engenharia de Software da IEEE [Gse90], por exemplo, não apresenta a definição do termo “análise de impacto”.

Apesar de não haver um consenso formal, alguns autores apresentam suas definições. [Rad86] definiu como a ação de examinar um impacto para determinar seus efeitos, isto é, representa a avaliação do efeito ou do resultado de fazer uma alteração em um sistema ou software.

Pfleeger definiu análise de impacto como a avaliação de muitos riscos associados a uma mudança, incluindo estimativas dos efeitos nos recursos, esforço e cronograma [Pfl90]. O autor estende aqui a definição a um contexto mais amplo de desenvolvimento de software e não apenas a alterações em código fonte.

Evoluindo nesta definição, Arnold e Bohner [Arn96] descrevem:

> A análise de impacto corresponde à identificação de potenciais consequências de uma mudança ou a estimativa do que precisa ser modificado com o objetivo de realizar uma mudança, incluindo estimativas de custos e cronogramas [Arn96].

Pela definição dos autores, identifica-se que a análise de impacto é utilizada para estimar o que será afetado no software e nas documentações associadas em virtude de uma requisição de mudança. As informações desta análise podem ser utilizadas para alterar o plano de desenvolvimento, realizar mudanças em software bem como rastrear efeitos dessas mudanças. A análise de impacto fornece a visibilidade desses potenciais efeitos antes que mudanças sejam implementadas, podendo assim tornar mais fácil e precisa sua implementação. Diante do exposto e pela sua relevância, utilizaremos a definição de Arnold e Bohner como referência para este trabalho.

A análise de impacto é frequentemente utilizada para auditar os efeitos de uma mudança em um sistema após a sua implementação, porém abordagens mais proativas utilizam a análise de impacto para prever os efeitos de uma mudança antes que a mesma seja executada [Arn96]. Neste caso, a análise de impacto precede ou é utilizada em conjunto com uma determinada mudança, fornecendo as entradas necessárias para sua implementação. Assim, a análise de impacto fornece o entendimento do escopo do que deveria ser alterado em um software devido a uma solicitação de mudança [Arn93]. Por exemplo, o time de desenvolvimento pode considerar diferentes abordagens para implementar uma mudança e escolher aquela que tiver menor impacto com relação a custos ou prazos. Com isso, torna-se possível planejar a mudança ao invés de lidar com suas consequências.

A falta de uma definição comum do termo análise de impacto pode levar ao surgimento de alguns problemas, tais como os citados por Arnold e Bohner [Arn93]:

- dificuldade de avaliar o que engenheiros de software entendem como análise de impacto devido à ausência de definições explícitas;
- ausência de dimensões específicas para comparar diferentes abordagens de análise de impacto;
- dificuldade para identificar se existem informações suficientes disponíveis para comparações significativas;
- dificuldade para discernir quando diferentes trabalhos sobre análise de impacto se relacionam.

Apesar das limitações a respeito de uma definição comum, a prática propriamente dita vem sendo largamente empregada. Dentre as principais razões, Ajila [Aji95] descreve:

- estimar o custo de uma mudança, ou seja, se uma alteração irá afetar várias partes disjuntas do software, então talvez seja melhor não realizá-la ou reexaminá-la;
- entender a razão e o relacionamento entre o escopo da alteração e a estrutura do sistema. Neste caso, através do rastreamento dos artefatos que serão afetados é possível conhecer quais deles realmente serão modificados e quais têm chance de ser;
- registrar o histórico de uma mudança em uma ferramenta de gestão de configuração e avaliar se esta foi realizada com qualidade, não comprometendo outras funcionalidades do sistema;
- saber quais partes do software deverão ser realizados testes de regressão de forma a garantir a qualidade do sistema, principalmente nas suas seções críticas.

2.1.2. Evolução da Análise de Impacto

Em 1967, durante conferência de NATO[1], começou-se a se discutir a importância do software em diversos aspectos da sociedade, sugerindo que o desenvolvimento seja menos *ad-hoc* e fundamentado em teorias, padrões e disciplinas que visem obter software confiável, eficiente e economicamente viável [Nau69]. Entre as várias atividades relacionadas com a Engenharia de Software, a manutenção foi discutida como sendo a atividade executada após sua entrega e implantação. Esta atividade foi consolidada por Royce que propôs em 1970 o modelo em cascata para o desenvolvimento de software [Roy70], formalizando a atividade relacionada com a manutenção e evolução do software.

A prática de análise de impacto também começou a ser discutida durante o início da história da engenharia de software. Em 1972, Haney discute aspectos relacionados ao impacto de mudanças em software e propõe um modelo para propagação dos efeitos em um sistema chamado Análise de Conexão de Módulos (*Module Connection Analysis*)

[1] http://homepages.cs.ncl.ac.uk/brian.randell/NATO/N1969/index.html

[Han72]. Este modelo pressupõe que para qualquer par de módulos (que pode ser entendido como componente, subsistema, etc.), existe uma probabilidade finita de que a mudança em um primeiro módulo requer uma mudança em um segundo módulo, definindo assim uma matriz de conexão probabilística.

Pouco mais tarde, Yau propõe em 1978 uma técnica para analisar os efeitos colaterais de uma mudança e estimar a complexidade da modificação [Yau78]. A técnica proposta considera a perspectiva funcional, na qual avalia o efeito colateral sob a perspectiva lógica do programa utilizando a análise léxica do código e o grafo de invocação do programa, que representa um grafo de controle de fluxo baseado em blocos. Com base nesta análise léxica, Yau propõe um algoritmo que computa o efeito colateral através da propagação de um critério de mudança, representado por um conjunto de variáveis, sobre o grafo de invocação, sugerindo ao final quais são os módulos do programa impactados por este critério. A técnica apresentada pode ser utilizada para determinar como uma mudança em uma área do código fonte se propaga e causa mudanças em outras áreas.

Um ano mais tarde, Weiser apresenta uma técnica chamada particionamento de programa [Wei79], cujo objetivo é reduzir um programa em sua forma mínima produzindo o mesmo resultado. Esta técnica visa recuperar as partições executáveis que contém apenas o código na qual uma variável em particular depende. O particionamento, que será melhor explicado na Seção 2.1.5, pode ser utilizado para determinar as dependências no código, minimizando assim os efeitos colaterais. O conceito de rastreabilidade de software também não é algo recente e geralmente é discutido como o elo entre os requisitos do sistema e seus conceitos definidos pela disciplina de Gerenciamento de Configuração [Ber79]. A rastreabilidade e a análise de código representam a base dos modelos de análise de impacto.

No ano 2000, Bohner discute o problema das datas Y2K, que tornou evidente o esforço necessário para identificar as porções de código que necessitariam adaptações devido à virada do milênio [Boh96]. Esta necessidade serviu para alertar as organizações na época sobre a relevância de métodos de análise de impacto explícitos.

A análise de impacto, ainda hoje, possui como base as técnicas e estratégias fundamentadas há muito tempo e focadas na análise de código fonte, incluindo particionamento e análise de efeitos colaterais. Atualmente é possível identificar a evolução dessas técnicas devido ao amadurecimento da Engenharia de Software e sua colaboração com demais disciplinas da Ciência da Computação. Técnicas como

Recuperação de Informação, Processamento de Linguagem Natural, Representação do Conhecimento, Redes Semânticas, entre outras, vêm sendo empregadas em conjunto com técnicas tradicionais para avaliar o impacto resultante de determinada mudança conforme apresentado nas Seções 2.2.4 e 2.3.5. Esta tendência promove a abstração das estruturas internas do código, tais como variáveis e controle de dependência, para o nível conceitual, conforme será discutido na revisão sistemática apresentada na Seção 2.4.

2.1.3. Estratégias para Análise de Impacto

Existem diversas estratégias para analisar o impacto de determinada mudança no software, com diferentes níveis de automação. Arnold e Bohner apresentam algumas dessas estratégias [Arn93]:

- navegação por um programa, abrindo e fechando arquivos relacionados;
- pesquisa por especificações de análise e projeto para determinar o escopo da mudança;
- utilização de um sistema de gerenciamento de configurações para mapear e procurar mudanças;
- entrevista com desenvolvedores que possuem conhecimento da aplicação;
- análise de rastreabilidade para identificar artefatos a serem modificados;
- análise de dependência pelo particionamento de programas para verificar o subconjunto desse programa que pode ser afetado pelo valor de uma dada variável [Gal91].

Observa-se que estas estratégias podem ser manuais e automatizadas por algoritmos. Pode-se argumentar que algumas estratégias pertencem as duas categorias, porém neste trabalho as quatro primeiras serão consideradas manuais enquanto as duas últimas automatizadas. As estratégias manuais geralmente são desempenhadas por humanos e não necessitam de uma infraestrutura específica, enquanto as automatizadas necessitam dessa infraestrutura para rastrear objetos e estimar o impacto de uma mudança.

As estratégias manuais são discutidas por Jönsson [Jon07] que argumenta que as mesmas não são fortemente dependentes de especificações estruturadas. Consequentemente, existe o risco de serem menos precisas na predição de impactos. Por outro lado, são mais fáceis de serem introduzidas em um processo de gerenciamento de

mudanças e são geralmente empregadas na indústria sem grande preocupação quanto a sua precisão. Estas estratégias são geralmente empregadas utilizando documentação de projeto e entrevistas para identificar as entidades primariamente impactadas.

As documentações de projeto podem considerar diferentes artefatos, como diagramas UML, descrições textuais de software e componentes, etc., e o seu sucesso depende de diferentes fatores, como o conhecimento e habilidade do responsável pela análise e a disponibilidade e confiabilidade da documentação.

Lindvall em [Lin98] apresenta que entrevistas com desenvolvedores que possuem conhecimento sobre o sistema representam provavelmente a forma mais comum de se estimar impacto e são mais efetivas do que a pesquisa em documentos e outras fontes. A segunda forma mais comum de análise de impacto é a análise do código fonte. No estudo apresentado, todos os participantes do experimento que realizaram a análise de impacto entrevistaram desenvolvedores e consultaram o código fonte, porém apenas metade consultaram documentações como casos de uso e modelos de objetos. Quando questionados sobre os métodos de análise, os participantes argumentaram que a documentação não abrangia todos os detalhes necessários ou não estava atualizada, enquanto “o código fonte, por sua vez, está sempre atualizado”.

Estratégias de análise de impacto automatizadas geralmente empregam algoritmos para identificar a propagação de mudanças e seus impactos indiretos. Para tanto, possuem como pré-requisito grafos de relacionamento [Jon07], que constituem uma informação semântica que evidencia as associações entre entidades computacionais. Estratégias como análise de rastreabilidade, dependência e particionamento são geralmente utilizadas para identificar os impactos de mudanças e serão apresentadas em detalhes nas próximas subseções.

A primeira estratégia para a análise de impacto que pode ser automatizada é a identificação do relacionamento entre entidades de software. Essa etapa permite que as relações sejam estabelecidas, por exemplo, entre o código fonte e requisitos de software. A segunda estratégia automatizada se preocupa em analisar o relacionamento entre módulos de um programa, baseados em objetos, fluxo de dados, variáveis, invocação, etc.

A diferença entre a análise de rastreabilidade e análise de impacto de uma mudança é bastante tênue, mas significativa. A rastreabilidade preocupa-se em definir a relação existente entre os artefatos do software. A análise de impacto, por sua vez,

determina quais ações devem ser tomadas para realizar a alteração, podendo estimar quais serão as consequências desta. Apesar da diferença, ambas as abordagens estão intimamente relacionadas, uma vez que a metodologia utilizada para analisar o impacto é baseada na forma como a rastreabilidade foi construída. Neste sentido, a análise de impacto pode ser dividida em duas etapas: análise de rastreabilidade e análise de dependência [Nea02].

Fundamentalmente a diferença entre estas duas estratégias é o nível de detalhe: análise de rastreabilidade se preocupa em analisar as associações entre todos os tipos de entidades de software, tais como código fonte, requisitos, casos de teste, etc., enquanto a análise de dependência analisa em baixo nível as dependências extraídas do código fonte [Pfl90].

2.1.4. Análise de Rastreabilidade

Rastreabilidade é a capacidade de inter-relacionar e recuperar entidades que se identificam durante um processo. Segundo o Glossário de Engenharia de Software da IEEE [Gse90], rastreabilidade é:

> (1) o grau no qual relacionamentos podem ser estabelecidos entre dois ou mais produtos de um desenvolvimento de software, especialmente quando existem relações de predecessor e sucessor ou generalização e especialização entre esses produtos; por exemplo, o grau em que os requisitos e o projeto de um determinado componente de software se associam.
>
> (2) O grau em que cada elemento em um produto do desenvolvimento de software estabelece para sua razão de existir; por exemplo, o grau em que cada elemento em um gráfico de bolhas referencia os requisitos que o satisfaz.

Análise de rastreabilidade se refere à habilidade de definir e recuperar associações entre entidades, incluindo requisitos ou outros tipos de artefatos de software. A rastreabilidade de informações é representada por elos que relacionam requisitos a (1) outros requisitos, (2) artefatos que satisfazem esses requisitos (elos de satisfação) ou as suas fontes [Kas04].

Um ponto fundamental para rastreabilidade consiste em perceber um sistema de informação como um conjunto de documentos e artefatos que descrevem o software em diferentes níveis de abstração e não apenas como uma implementação. Neste contexto, Pfleeger define rastreabilidade como a habilidade de relacionar os itens dependentes dentro de um modelo e seus correspondentes em diferentes modelos [Pfl90]. Adicionalmente, esta técnica provê um apoio essencial para o entendimento da organização estrutural entre os requisitos e seus artefatos derivados [Pal00]. O sucesso

da análise de rastreabilidade depende fortemente da consistência e completeza dos relacionamentos identificados.

A rastreabilidade possui diversas classificações bem aceitas na literatura para avaliar a qualidade e o alcance dos relacionamentos. A primeira dimensão de rastreabilidade diz respeito à interconexão dos itens relativos aos modelos. Nesta perspectiva, Pfleeger apresenta os seguintes tipos [Pfl90]:

- rastreabilidade vertical: os itens relacionados pertencem ao mesmo artefato;
- rastreabilidade horizontal: os itens relacionados pertencem a artefatos diferentes.

Ambos os tipos de rastreabilidade são necessários para compreender um conjunto completo de relacionamentos pertinentes à análise de impacto.

Para o mapeamento, tanto vertical quanto horizontal, Pfleeger sugere o uso de grafos, onde um conjunto de objetos corresponde aos nodos e os relacionamentos correspondem a arestas [Pfl90]. É sugerida uma modularização entre requisitos, projeto, código e teste que estruturam uma organização vertical de seus componentes, e estes componentes se relacionam horizontalmente com os demais módulos. A Figura 2.1 apresenta a proposta de organização horizontal entre componentes de software.

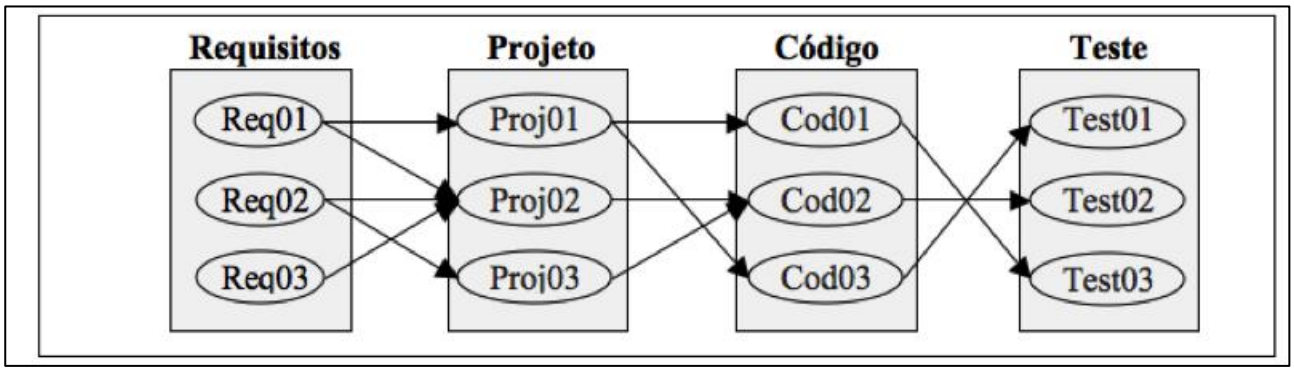


Figura 2.1 – Rastreabilidade em Artefatos de Software segundo (proposto em [Pfl90]).

Segundo Gotel e Finkelstein, um termo mais comum para rastreabilidade é apresentado como Rastreabilidade de Requisitos que corresponde à habilidade de relacionar requisitos específicos com suas implementações [Got94]. Nesta perspectiva, é apresentada uma divisão de rastreabilidade em duas dimensões principais:

- pré-rastreabilidade: capacidade de rastrear, a partir dos requisitos, as necessidades dos usuários;

- pós-rastreabilidade: capacidade de recuperar como cada requisito é implementado.

Lindvall, em [Lin94], apresenta duas classificações para rastreabilidade levando em conta o tipo de relacionamento entre os elementos:

- relacionamento explícito: representa a capacidade de recuperar relacionamentos entre dois modelos utilizando elos previamente definidos;
- relacionamento implícito: em oposição ao relacionamento explícito, representam associações organizadas por convenções como, por exemplo, padronização para nomes de entidades.

Uma abordagem explorada por Bianchi et al. [Bia00] diz respeito a relações obtidas através de sintaxe e semântica dos modelos que consistem em:

- relacionamentos estruturais: que corresponde a rastreabilidade associada a uma dependência sintática entre alguns artefatos originados a partir do código fonte;
- relacionamentos cognitivos: que dizem respeito aos elos de rastreabilidade obtidos através da semântica associada às decisões de projeto [Rug90].

Uma abordagem comum para manter elos de rastreabilidade é o uso da matriz de rastreabilidade [Kot98, Lef99]. Uma matriz de rastreabilidade é uma matriz em que cada linha e cada coluna correspondem a uma entidade de software como, por exemplo, um requisito. A relação entre as duas entidades é expressa colocando uma marca entre a intersecção da linha e coluna. Por exemplo, o relacionamento entre um requisito, organizado em uma linha da matriz, e uma classe do software, representado por uma coluna, pode ser representado por um símbolo de intersecção na matriz entre a linha e coluna associada. Este símbolo representa que o requisito é implementado parcialmente ou totalmente por aquela a classe.

Lindvall define que rastreabilidade é uma propriedade de um sistema e um fator de qualidade que corresponde ao entendimento do software e ao acompanhamento da documentação de seus modelos [Lin94]. Apesar disso, Ramesh e Jarke [Ram01] apresentam um estudo que evidencia que as práticas de rastreabilidade não são em geral completas, pois não endereçam a informação semântica (*rationale*) destes elos de rastreabilidade. Um relacionamento entre um requisito e uma classe, por exemplo, pode resultar em interpretações diferentes para interessados diferentes. Ramesh e Jarke

argumentam que, por exemplo, o relacionamento entre um requisito e uma classe pode ser interpretado como que a “classe implementa o requisito” ou que a “classe representa uma restrição para o requisito“. Um elo de rastreabilidade simples representa apenas uma associação entre entidades, isto é, representa que uma entidade afeta outra, sem incluir a semântica que motivou essa associação. O fator de qualidade definido por Lindvall necessita de um suporte semântico para capturar o entendimento e razão que motivaram o elo de rastreabilidade.

De posse de um sistema rastreável, essa análise pode apoiar o desenvolvimento e manutenção do software, permitindo assim uma compreensão do sistema, apoiando a análise de impacto de mudanças. Neste caso, uma forma tradicional de estimar o impacto de mudanças é utilizar informações estáticas sobre o código, isto é, dependências, para identificar partes do código potencialmente afetadas.

2.1.5. Análise de Dependência

Segundo Law e Rothermel, as três principais técnicas de análise de impacto baseada em dependências são grafos de chamadas, particionamento estático e particionamento dinâmico [Law03].

Um grafo de chamadas é um grafo direto no qual os nodos representam funções (ou métodos) e uma aresta transitiva entre A e B significa que A pode chamar B [Ors03]. Através da definição de dependência direta entre os métodos, é possível avaliar o impacto pela manutenção de uma função específica.

O particionamento de um programa consiste em reduzir esse programa para a sua forma mínima, mantendo ainda seu comportamento inicial [Wei81]. Por exemplo, inserindo mudanças no código e aplicando particionamento, que se baseia na construção de um grafo de dependência entre diversas entidades computacionais para determinar a dependência transitiva de determinada entidade, permite-se identificar um conjunto de pontos de impacto decorrentes de determinada mudança. A ideia por traz do particionamento é justamente identificar apenas o código de interesse para determinado comportamento para, então, modificar apenas essa porção de código. Por exemplo, considere a Figura 2.2.

A Figura 2.2(a) apresenta um exemplo de código que lê um número n e computa a soma e o produto dos primeiros n números positivos. A Figura 2.2(b) representa apenas uma porção do código relacionada ao cálculo do produto, ignorando todos os demais

trechos do código que não dizem respeito a essa operação. O particionamento estático realiza a identificação do conjunto de todas as declarações transitivas relativas a determinado fluxo de controle ou de dados, considerando apenas as informações estáticas presentes no código. O particionamento estático reduz o número de análises se comparado ao grafo de chamadas, pois considera apenas os valores potencialmente afetados em algum ponto de interesse, também chamado de critério de particionamento [Tip95]. Critérios de particionamento geralmente são definidos em uma parte de um programa contendo a ocorrência, por exemplo, a linha 10, *write(product)*, da Figura 2.2, e uma variável, por exemplo, a variável *product* da Figura 2.2. Neste exemplo, o critério de particionamento é (10, *product*) e o código resultante é representado pela Figura 2.2 (b). Neste exemplo, qualquer referência a variável *sum* é considerada fora do escopo da partição analisada.

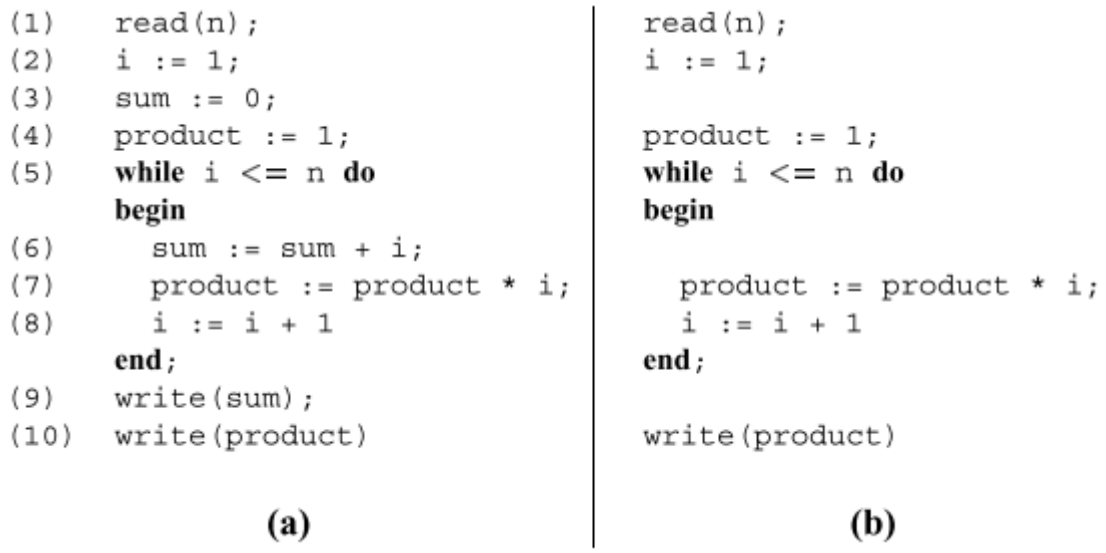


Figura 2.2 – (a) Um exemplo de programa. (b) O particionamento estático do programa segundo o critério de particionamento (10, product) (proposto em [Tip95]).

Dentre os particionamentos, o estático é computado sem realizar suposições sobre as entradas de um programa enquanto o dinâmico utiliza casos de testes específicos em sua definição para determinar suas possíveis entradas [Tip95]. Neste caso, apenas as dependências que ocorrem em uma execução específica (caso de teste) devem ser levadas em consideração. Tipicamente, o critério de particionamento dinâmico é composto por três elementos: entrada do programa, ocorrência e variável. A ocorrência deve levar em consideração dados históricos, isto é, o número de vezes que esta declaração ocorreu no código.

Por exemplo, a Figura 2.3 apresenta um exemplo de programa e o critério de particionamento. Esse critério é definido por (n=2, 81, x), sendo que n=2 representa a entrada, 81 representa a primeira ocorrência da declaração e x a variável de interesse. Note que como n=2, o loop será executado duas vezes passando uma única vez pelas declarações x:= 17 e x:=18. A linha 6 na Figura 2.3 (b) está omitida, pois a variável x na primeira iteração do loop é ignorada devido à atribuição do valor 17 da segunda iteração.

```
(1)   read(n);
(2)   i := 1;
(3)   while (i <= n) do
      begin
(4)     if (i mod 2 = 0) then
(5)       x := 17
        else
(6)       x := 18;
(7)     i := i + 1
      end;
(8)   write(x)
```

(a)

```
read(n);
i := 1;
while (i <= n) do
begin
  if (i mod 2 = 0) then
    x := 17
  else
        ;
  i := i + 1
end;
write(x)
```

(b)

Figura 2.3 – (a) Exemplo de programa. (b) Particionamento dinâmico usando o critério (n=2, 81, x) (proposto em [Tip95]).

No exemplo da Figura 2.3, o particionamento estático, cujo critério é (8, x), irá considerar todo o algoritmo, isto é, a Figura 2.3 (a), enquanto o particionamento dinâmico irá considerar apenas a partição representada pela Figura 2.3 (b).

Para o particionamento entre procedimentos, existem algumas técnicas comuns, como o Grafo de Fluxo de Controle (Figura 2.4), que representa a ordem de invocação das instruções, o Grafo de Dependência (Figura 2.5), que representa a subordinação das estruturas, e Grafo de Fluxo de Dados (Figura 2.6), que representa o uso de cada variável ao longo do programa.

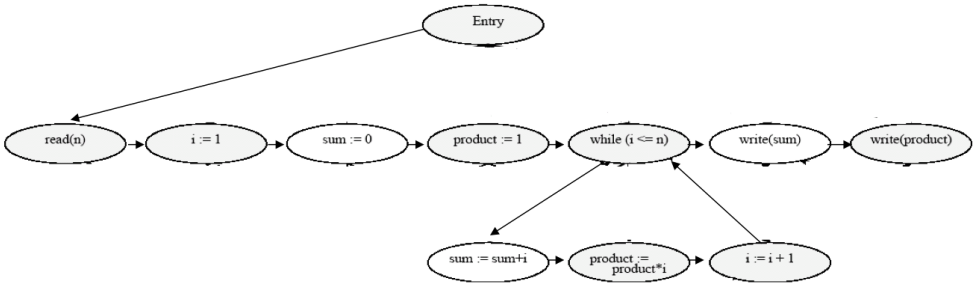


Figura 2.4 – Grafo de Fluxo de Controle (proposto em [Tip95]).

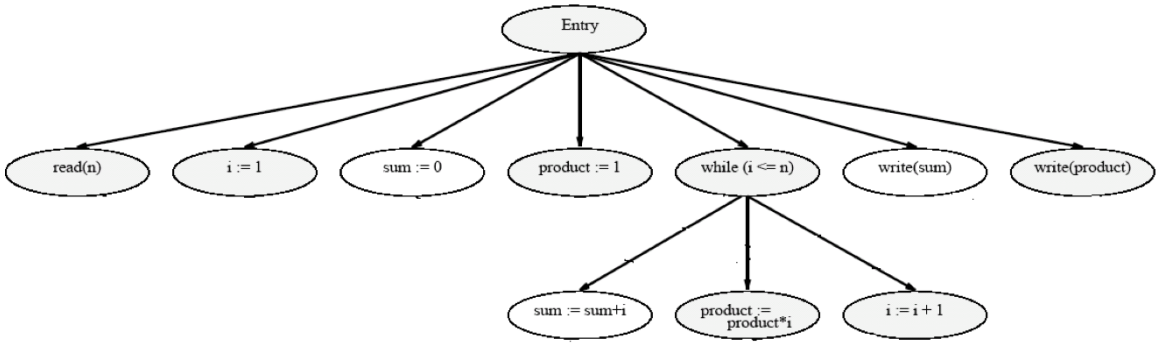


Figura 2.5 – Grafo de Dependência (proposto em [Tip95]).

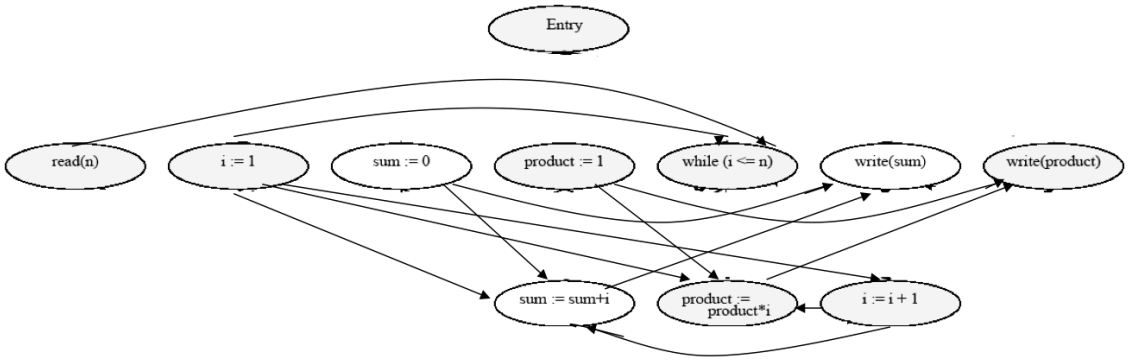


Figura 2.6 – Grafo de Fluxo de Dados (proposto em [Tip95]).

Segundo Law e Rothermel, cada uma dessas abordagens para análise de dependência tem suas vantagens e desvantagens [Law03]:

- o fechamento de transitividade em grafos de chamadas é relativamente barato, porém pode ser altamente propenso a erro, identificando impactos onde não existem e falhando na identificação de impactos onde existem;
- o particionamento estático pode prever impactos de mudanças em um modelo mais conservador e seguro, entretanto ele foca em todos os possíveis comportamentos do programa podendo retornar um conjunto de elementos impactados muito grande ou impreciso comparado àqueles perfis operacionais esperados por um sistema e útil ao time de desenvolvimento;
- o particionamento dinâmico pode prever um impacto relativo de um programa específico ou perfil operacional, o qual pode ser útil para as tarefas de manutenção do software, sacrificando, porém a segurança na auditoria dos impactos resultantes;
- tanto o particionamento estático quanto dinâmico são relativamente caros em termos computacionais, possuindo dependência de dados, de controle e

análise de *aliases*. Nesta perspectiva, a análise por grafo de chamadas é mais barata;

- todas as três abordagens baseiam-se no código fonte para determinar a invocação das estruturas ou dependências estáticas no código e requerem um esforço computacional significativo para reprocessar toda a informação necessária para avaliar o impacto em entregas subsequentes do código fonte do sistema em manutenção.

O particionamento em um grafo de chamadas pode gerar estimativas inseguras sobre o conjunto de operações afetadas pelas mudanças [Arn96], gerando resultados imprecisos e superestimando os efeitos de mudanças [Ors03]. Recentemente, pesquisadores vêm considerando formas mais precisas de auditar os impactos utilizando informações capturadas durante a execução dessas mudanças.

Por exemplo, Law e Rothermel definiram uma técnica para análise de impacto que utiliza uma técnica chamada caminho completo de programas [Law03]. Esta técnica captura a dinâmica de um programa através de seus fluxos de controle e, em oposição às técnicas comuns de perfis de caminho que registram caminhos intraprocedurais e acíclicos, esta proposta captura os fluxos completos de programas incluindo ciclos e interações interprocedurais. Nesta abordagem, se um procedimento *p* é alterado, todos os procedimentos que são invocados após *p*, bem como qualquer procedimento que está na pilha de chamada aguardando o retorno deste procedimento, são incluídos no conjunto de procedimentos potencialmente impactados.

Embora técnicas baseadas no rastreamento de execução possam atingir melhores resultados do que técnicas de análise de impacto tradicionais [Law03], tais como grafo de chamadas e particionamento de programas estático e dinâmico, essas técnicas são restringidas pela qualidade dos dados que são utilizados. Como restrições, tem-se que (1) a quantidade de rastros de execução utilizados pode crescer exponencialmente, e (2) algoritmos que comprimem estes rastros a um nível computacionalmente razoável podem se tornar custosos, comprometendo o seu uso em tempo de execução na medida em que os rastros são produzidos [Ors03].

Conforme apresentado, a análise de impacto corresponde à identificação de potenciais consequências de determinada mudança ou a estimativa do trabalho necessário para completar essa mudança. No que se referem a impacto, duas perspectivas são pertinentes: (1) perspectiva de negócio, relacionada às preocupações

das partes interessadas em verificar se as regras de negócio existentes são impactadas por uma mudança, necessitando adequação; (2) perspectiva do software, relacionada com as alterações no código fonte e documentação associada. Oliveira, em [Oli10], apresenta um estudo sobre diferentes técnicas de rastreabilidade e evidencia que nenhuma das avaliadas lida diretamente com a perspectiva de negócio com razoável nível de precisão para os impactos descobertos. Para ambas as perspectivas, dois tipos de medidas são consideradas importantes para a avaliação dos impactos: precisão, para medir a exatidão ou fidelidade, e revocação, como uma medida de completude [Ols08].

2.1.6. Evolução de Software

A análise de impacto é uma atividade fundamental para a evolução de software. Essa evolução corresponde à etapa posterior a primeira implantação do sistema em produção, na qual novas necessidades são identificadas.

A evolução do software é inevitável, pois, conforme definido pela primeira lei de Lehman, chamada Mudança Contínua, um programa usado em um ambiente real deve necessariamente mudar ou irá se tornar progressivamente menos útil [Leh96]. Isso se deve em contextos nos quais as funcionalidades oferecidas pelos sistemas requerem evolução contínua para manter a satisfação do usuário.

Como evidência da importância da evolução do software, estudos como apresentados por Erlikh [Erl00] sugerem que 90% dos custos do software estão em sua evolução. De forma geral, para sistemas encomendados e específicos de domínio, os custos de manutenção geralmente excedem os custos de desenvolvimento em uma proporção de 80% para 20% [Pfl04].

A evolução do software é direcionada por propostas de mudanças. Essas propostas podem envolver requisitos não implementados, solicitação de novos requisitos ou reparos. Este processo inclui a análise de mudanças, planejamento de entregas, implementação e liberação do software, considerando também o custo e impacto dessas mudanças. Para ilustrar o processo de evolução de software, a Figura 2.4 apresenta o processo de evolução de sistemas proposto por Sommerville [Som09].

A Figura 2.7 ilustra que após a solicitação de mudanças, é necessário realizar uma análise do impacto dessa solicitação para então realizar um planejamento, implementação e liberação do sistema. Esse processo varia consideravelmente entre organizações, desde um processo informal decorrente de conversas entre usuários e

desenvolvedores, até um processo formalizado com documentação estruturada e produzida em estágios definidos, chamado de gerenciamento de mudanças.

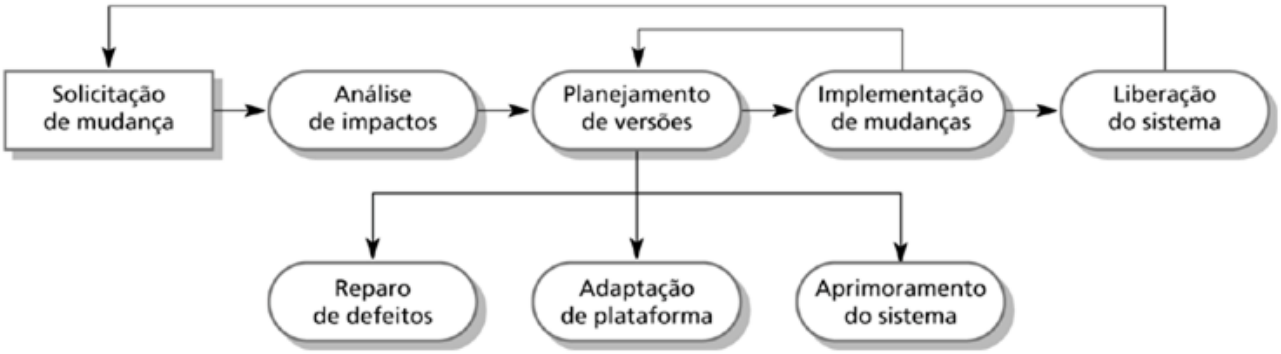


Figura 2.7 – Processo de evolução de sistema (proposto em [Som09]).

O gerenciamento de mudanças no contexto do desenvolvimento de software representa o processo de solicitar, analisar a viabilidade, planejar, implementar e avaliar as mudanças em um sistema. A finalidade de ter um processo padrão de controle de mudanças é assegurar que as mudanças feitas em um projeto sejam consistentes e que as partes interessadas sejam informadas do estado do produto, das mudanças feitas nele e do impacto de custo e cronograma gerados por essas mudanças [Kru03].

Conforme Sommerville, procedimentos de gerenciamento de mudanças dizem respeito à análise de custo e ao benefício de mudanças propostas, à aprovação de mudanças viáveis e à rastreabilidade de quais componentes de software foram alterados [Som09].

Conforme o Kruchten, a primeira etapa desse processo consiste em preencher um formulário formal de solicitação de mudanças chamado Formulário de Requisição de Mudança (RdM) que descreve as informações sobre a mudança, tais como finalidade, breve resumo, ocorrência, responsabilidade e adaptação [Kru03].

Neste formulário deve ser inicialmente descrita a mudança em uma perspectiva de negócios para que, uma vez validada e aprovada, seja encaminhada para a equipe técnica avaliar o impacto dessa solicitação no restante do sistema, identificando os componentes afetados por esta mudança. Para essa análise, são utilizadas informações do banco de dados de configurações e do código fonte do software. Após a análise de impacto, essa mudança é avaliada e o custo para realização é estimado [Som09].

Após esse levantamento, o RdM é encaminhado para um comitê de controle de mudanças, que corresponde a um grupo formalmente constituído de partes interessadas responsáveis pela revisão, avaliação, aprovação, atraso ou rejeição de mudanças feitas

em um projeto, com registro de todas as decisões e recomendações [Pmb08]. Esse comitê considera o impacto mais sobre o ponto de vista técnico do que estratégico e organizacional [Som09].

A evolução do software é consequência de mudanças das necessidades de negócio que, muitas vezes, refletem uma evolução do conhecimento sobre o domínio da aplicação. O conhecimento a cerca do domínio é geralmente capturado através de produtos de trabalho, tais como especificações funcionais, diagramas e código fonte. Todos esses artefatos representam uma perspectiva de conhecimento sobre o sistema em um determinado período.

Um fator crítico de sucesso durante as etapas de desenvolvimento e manutenção de software é a comunicação e colaborações do time para definir uma visão comum sobre o que é e como deve se comportar o sistema. Este conhecimento compartilhado desempenha um papel fundamental para nivelar o entendimento entre as partes interessadas sobre quais são os requisitos de negócios e as entidades computacionais derivadas desses requisitos.

2.2. Ontologias e Análise de Impacto

2.2.1. Definições de Ontologia

O termo “ontologia” surgiu na filosofia através de Aristóteles e significa “uma explicação sistemática da existência”, isto é, a definição de um domínio do conhecimento em um nível genérico, utilizada para especificar o que existe ou o que se pode dizer sobre o mundo.

Em Ciência da Computação, ontologias representam à aquisição do conhecimento a partir de dados semiestruturados utilizando um conjunto de métodos, técnicas ou processos automáticos ou semiautomáticos. Dentro de Ciência da Computação, o termo “ontologia” teve sua origem na comunidade de Inteligência Artificial.

Segundo Thomas Gruber, aquilo que existe é aquilo que pode ser representado [Gru93]. Quando o conhecimento sobre um domínio é especificado em uma linguagem declarativa, o conjunto de objetos que pode ser representado é denominado universo de discurso. Pode-se descrever uma ontologia de um programa pela definição de um conjunto representacional de termos. As definições associam os nomes de entidades do universo de discurso (como classes, relacionamentos, funções ou outros objetos) com um

texto que descreve o que os nomes significam e com os axiomas formais, que tanto restringem a interpretação destes termos quanto a boa formação no seu uso.

Ontologias são utilizadas para definir os termos usados para descrever e representar um domínio de informação. Entende-se por domínio de informação uma específica área de conhecimento. Para tanto, ontologias incluem definições de conceitos e seus relacionamentos, tornando assim o conhecimento compartilhado e, de certa forma, reusável.

Gruber, em [Gru95], define ontologias como uma “especificação explícita de uma conceituação”. Uma conceituação é uma abstração simplificada da realidade que se deseja representar, isto é, um conjunto de objetos, restrições, relacionamentos e entidades que se assumem necessárias em alguma área de aplicação.

A conceituação de Gruber foi modificada por Borst, em [Bor97], definindo ontologias como uma “especificação formal de uma conceituação compartilhada”. Esta definição enfatiza o fato que deve haver um acordo na conceituação do que é especificado.

Conforme Nino Cocchiarella [Coc01], ontologias formais representam o desenvolvimento sistemático, formal e axiomático da lógica de todos os modos da existência. Ela define as propriedades formais, a classificação de entidades no mundo (objetos físicos, eventos, etc.) e as categorias que modelam o mundo (conceitos, propriedades, etc.).

Quanto à categorização de ontologias, Guarino [Gua98] classifica ontologias com relação ao seu nível de dependência a uma tarefa particular ou ponto de vista. Guarino distinguiu as seguintes ontologias: de alto nível, de domínio, de tarefa e de aplicação. Ontologias de alto nível descrevem conceitos gerais como espaço, tempo, objetos, eventos, ações, etc., que são independentes de um problema particular. Ontologias de domínio e de tarefa descrevem, respectivamente, o vocabulário genérico de um domínio (como medicina, automóveis) ou uma tarefa ou atividade em particular (como diagnosticar ou vender). Ontologias da aplicação descrevem os conceitos de um domínio ou tarefa particular, correspondendo aos papeis executados por entidades do domínio enquanto executando certa atividade.

2.2.2. Aprendizagem e População de Ontologia

O uso de ontologias pode fornecer benefícios potenciais para várias aplicações, porém é sabido que sua construção é custosa e geralmente referenciada como um gargalo de aquisição de conhecimento [Mae02]. Neste contexto, diferentes técnicas e métodos para criar e manter ontologias vêm sendo propostas. A engenharia de ontologia se refere ao conjunto de atividades relacionadas ao processo de desenvolvimento de ontologias, seu ciclo de vida e as metodologias, ferramentas e linguagens para sua construção [Gom04].

A modelagem de ontologias de domínios não triviais é uma tarefa de fato difícil e que necessita de bastante tempo e esforço [Mae02]. A compensação entre modelar uma grande quantidade de conhecimento e fornecer o máximo de abstrações possíveis para manter o modelo conciso faz com que a engenharia ontológica se torne uma tarefa desafiadora. Uma solução interessante é desenvolver uma abordagem para a aprendizagem automática da ontologia a partir de dados existentes.

O termo Aprendizagem de Ontologias (*ontology learning*) foi inicialmente definido por Maedche e Staab em [Mae02] e descrito como a aquisição de um modelo de domínio a partir de dados. Para que a aprendizagem de ontologias seja realizada, são necessários dados representativos para o domínio no qual a ontologia pretenda modelar. Estes dados podem ser, por exemplo, diagramas UML ou esquemas de banco de dados. O nome dado a essa aprendizagem de ontologias é *lifting* [Vol01] como basicamente consiste em mapear definições a partir do esquema para suas definições ontológicas. O aprendizado de ontologias também pode ser executado utilizando como fonte dados semiestruturados como documentos XML, HTML ou estruturas tabuladas [Piv05].

Cimiano em [Cim06] apresenta um conjunto de tarefas necessárias para a aprendizagem de ontologias, como a aquisição de terminologia relevante, identificação de termos sinônimos ou variações linguísticas, formação de conceitos e sua organização hierárquica, bem como a de propriedades, instanciação de esquemas e axiomas.

O processo de encontrar instâncias de conceitos e relacionamentos é comumente chamado de população de ontologias [Cim06], que representa a seleção de fragmentos de texto a partir de documentos para designá-los a conceitos da ontologia. Isso representa conectar conceitos e relacionamentos aos símbolos que são utilizados para referenciá-los e que, neste caso, implica na aquisição de conhecimento a partir do conhecimento linguístico sobre os termos que são utilizados para se referir a um conceito específico ou sinônimo potencial destes termos.

2.2.3. OWL – Web Ontology Language

Para se definir e manipular ontologias se sugere a utilização de linguagens que suportem estruturas para representação do conhecimento. Esta representação é realizada através da descrição formal de um conjunto de termos sobre um domínio específico. A definição de uma linguagem é necessária para a representação e descrição formal da estrutura que especifica uma conceituação.

Um termo que possui determinado significado pode variar sua semântica conforme o contexto empregado. Para solucionar este problema de ambiguidade e propor uma estrutura formal para a representação do conhecimento, surgiram algumas propostas de linguagens, como KIF [Kif98], XOL [Kar99], RDF [Rdf04], SHOE [Sho12], DAML [Dam12] e OWL [W3c12].

A linguagem OWL foi recomendada pela W3C [W3c12] em fevereiro de 2004 como linguagem para manipulação de ontologias e seu diferencial é a capacidade de processamento semântico através de inferência.

Uma semântica formal e o seu apoio lógico são geralmente providos através do mapeamento de uma linguagem ontológica em um formalismo. Estes requisitos foram a base para uma divisão da linguagem OWL em três sublinguagens: OWL Full, OWL DL e OWL Lite.

A linguagem OWL é construída sobre RDF e RDF Schema e baseada na sintaxe XML. O modelo básico de dados do RDF, e herdado por OWL, é definido através de:

- recurso (*resource*): qualquer entidade referenciada através de um URI (*Universal Resource Identifier*);
- propriedade (*property*): representam recursos, características que representam recursos ou relacionamento entre recursos;
- declaração (*statement*): corresponde a uma propriedade ou valor dessa propriedade associada a um recurso específico. Uma declaração é dividida em três partes: sujeito (recurso), predicado (propriedade do recurso) e objeto (valor da propriedade).

Para se estruturar um documento OWL, define-se em alto nível:

- classes: conjunto de instâncias com características comuns, podendo ser consideradas superclasses, relacionamentos e disjunções.

- propriedades: representam tipos (*datatypeProperties*), que identificam os valores primitivos das instâncias, como integer, float, string, boolean, etc.; objetos (*objectProperties*), que representam o relacionamento de duas instâncias; inversa (*inverseOf*), que representam um relacionamento bidirecional; e transitivas (*TransitiveProperty*), usado principalmente em relações de transitividade do tipo “parte de”.
- indivíduos: representam os objetos em um domínio, isto é, instâncias específicas. Verifica-se aqui que dois nomes podem representar o mesmo objeto no mundo real.

2.2.4. Ontologia Aplicada a Rastreabilidade de Software

Em [Nol07a, Nol07b] é proposta uma abordagem de rastreabilidade orientada a ontologias e integrada ao Processo Unificado [Kru03]. O objetivo dessa proposta é relacionar elementos que compõem certos diagramas UML com os conceitos de seu domínio de aplicação, utilizando para isso uma ontologia como o principal artefato para representação do conhecimento. O objetivo dessa integração é adicionar uma camada semântica entre diferentes modelos de software, apoiando assim os desenvolvedores na busca de informação relevante entre elementos de diagramas da UML.

Com o objetivo de formalizar a proposta de rastreabilidade orientada a ontologia, os seguintes objetivos específicos foram desenvolvidos:

- necessidade de definir uma abordagem sistemática para projetar e manter uma ontologia de aplicação no Processo Unificado;
- necessidade de definir uma estrutura ontológica para associar apropriadamente os conceitos do domínio com os artefatos de software;
- necessidade de uma abordagem sistemática para mapear os conceitos presentes em diferentes artefatos de desenvolvimento com os conceitos capturados pela ontologia da aplicação;
- necessidade de fornecer uma ferramenta apropriada para apoiar a rastreabilidade orientada a ontologias, estendendo uma ferramenta CASE existente.

Inicialmente foi proposta a criação de uma nova disciplina no Processo Unificado, chamada Modelagem do Conhecimento, a qual foi estruturada em três macro atividades:

Projeto, Manutenção e Verificação. A ontologia é projetada nas fases iniciais de desenvolvimento utilizando como base o Modelo de Domínio, documento este produzido durante a disciplina de Modelagem de Negócios. Este modelo utiliza a sintaxe do diagrama de classes da UML para representar os conceitos do mundo real. Com isso, a disciplina de Modelagem de Conhecimento refina a ontologia de domínio introduzindo novos relacionamentos que não podem ser expressos em diagramas de classes convencionais, como subPropertyOf. Durante a disciplina de Requisitos, novos conceitos e relacionamentos sobre o domínio vão surgindo, fornecendo assim insumos para refinar o Modelo de Domínio e a ontologia em desenvolvimento.

Uma vez que este modelo é considerado completo, os Engenheiros do Conhecimento (EC) [Mae02] podem iniciar suas atividades de manutenção. Esta macro atividade é responsável por manter a consistência entre o modelo de conhecimento gerado e a semântica dos artefatos produzidos durante o ciclo de vida de desenvolvimento. Além disso, o EC estabelece manualmente os elos de rastreabilidade entre conceitos do domínio e elementos UML.

Cada modificação na ontologia deve ser verificada pelas atividades descritas na macro atividade de Verificação. Estas atividades consistem em um processo de controle que analisa cada nova versão do modelo de conhecimento procurando por inconsistências.

Foi apresentada também uma estrutura ontológica chamada ONTrace responsável pela associação entre os recursos do domínio e os elementos dos diagramas UML. ONTrace representa um recurso descrito em uma classe OWL associada a uma propriedade (objectProperty) chamada ontraceRecover que mapeia elementos UML a conceitos do domínio. Este mapeamento inclui o relacionamento entre casos de uso, classes em um diagrama de classes e outros elementos descritos pela UML, a recursos de conhecimento, tais como classes e propriedades descritas em OWL. Para cada elo de rastreabilidade, uma instância da classe ONTrace é criada ou atualizada, mapeando algum conceito a elementos da UML.

A Figura 2.8 exemplifica um elo de rastreabilidade entre o caso de uso “UC01 – Manter Funcionário” e o conceito da ontologia “Funcionário” através da propriedade “ontraceRecover”. Adicionalmente, este elo especifica que este indivíduo é do tipo “UseCase”, que é um dos tipos definidos pela taxonomia estabelecida sobre a classe “Model”.

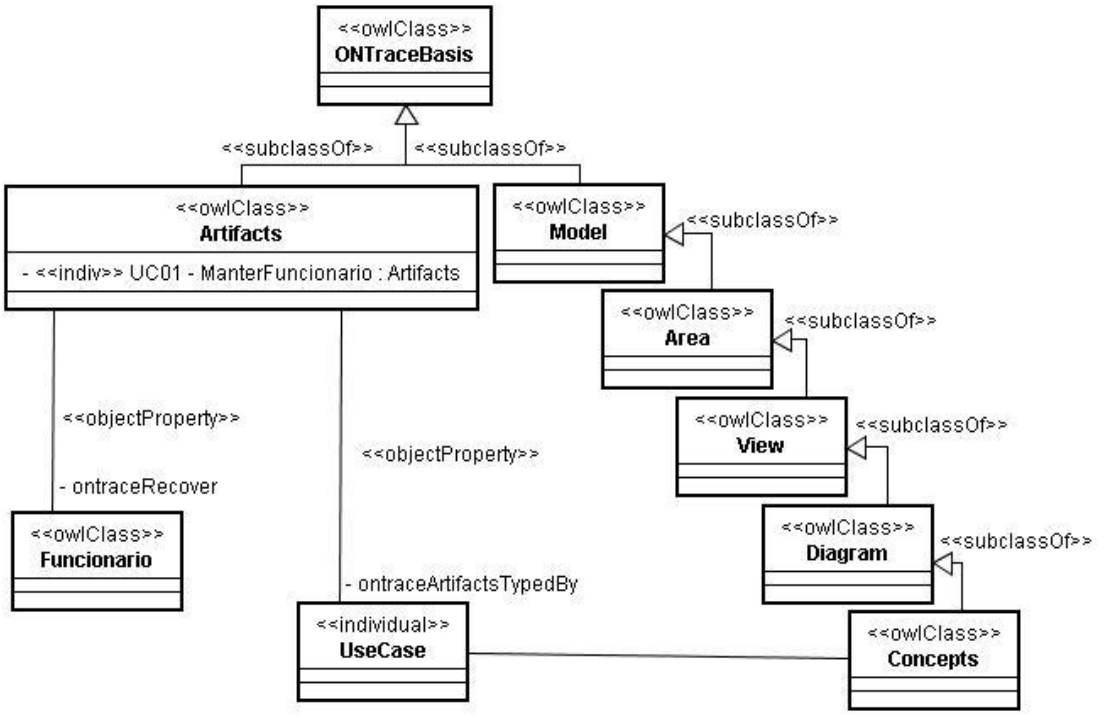


Figura 2.8 – Recurso ONTrace para rastreabilidade entre elementos UML.

Para ilustrar a rastreabilidade orientada a ontologias, considere um cenário de comércio eletrônico. Suponha que o caso de uso “Manter Cliente” está relacionado ao conceito da ontologia “Cliente”, enquanto outro caso de uso “Realizar Pedido” está relacionado aos conceitos “Cliente”, “Pedido” e “Produto”. Neste cenário, é possível inferir que “Manter Cliente” e “Realizar Pedido” se relacionam explicitamente através do conceito “Cliente” por um elo de rastreabilidade direto.

Neste mesmo exemplo, suponha que exista outro caso de uso chamado “Manter Funcionário”, que se relaciona com o conceito “Funcionário”. Existe também uma propriedade chamada “buscaInformacao” que relaciona os conceitos “Empregado” e “Cliente”. Neste cenário, é possível inferir um relacionamento implícito entre os artefatos “Manter Cliente” e “Manter Funcionário”, pois os conceitos que relacionam ambos estão associados pela propriedade “buscaInformacao”. É importante perceber que o uso de regras semânticas complexas em motores de inferência permite recuperar cenários muito menos óbvios que este exemplo, estendendo-se inclusive para diferentes produtos de trabalho.

Para apoiar esse trabalho, foi desenvolvida uma ferramenta chamada ONTrace IDE como uma extensão da ferramenta *open source* de modelagem ArgoUML [Arg12]. A ferramenta permite a extração da ontologia a partir do Modelo de Domínio, a geração de elos de rastreabilidade utilizando o recurso ONTrace da ontologia e a recuperação de elos

implícitos e explícitos. A Figura 2.9 apresenta a interface da ferramenta que permite recuperar elos de rastreabilidade explícitos e implícitos.

A avaliação da proposta foi feita por dois experimentos controlados aplicados a populações e em épocas diferentes. Este estudo empírico foi executado comparando a rastreabilidade orientada a ontologias com a tradicional rastreabilidade orientada a requisitos, considerando as perspectivas de esforço e precisão. O objetivo foi caracterizar a aplicabilidade e relevância dessa abordagem comparada a modelos amplamente empregados em empresas de desenvolvimento de software. Embora os resultados sejam limitados ao contexto dos experimentos, os mesmos parecem promissores em termos de aplicabilidade, mostrando um melhor desempenho nas perspectivas consideradas para a rastreabilidade orientada a ontologias tanto na primeira execução do experimento quanto em sua replicação. Maiores detalhes estão descritos em [Nol09].

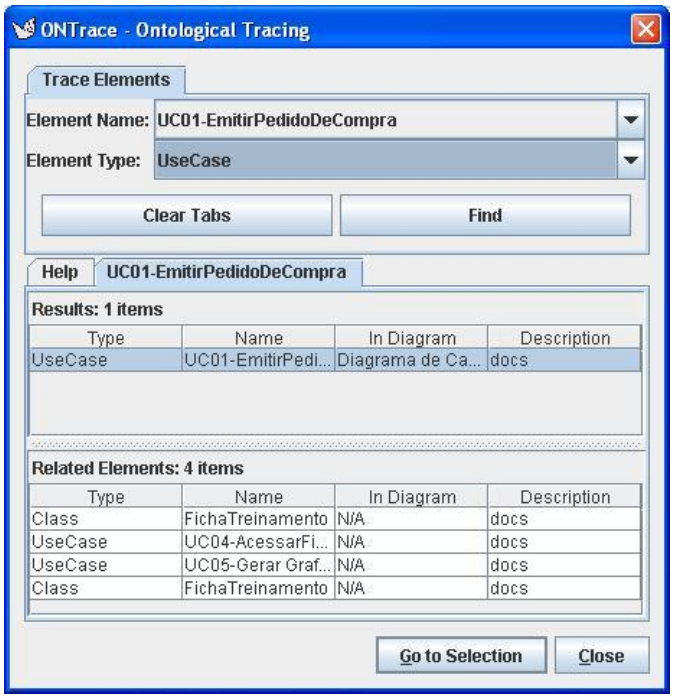


Figura 2.9 – Interface de busca da ferramenta ONTrace IDE.

2.2.5. Ontologia Aplicada a Análise de Impacto

Recentemente é possível perceber o crescimento do número de artigos científicos relacionados à integração de semântica formal nas metodologias de desenvolvimento de software. Diferentes aspectos do desenvolvimento vêm sendo melhorados pelo uso de técnicas de representação do conhecimento. Devido à similaridade entre ontologias e modelos orientados a objetos [Bat07], alguns pesquisadores vêm focando em como

melhorar o projeto e qualidade usando Lógica Descritiva, que é um subconjunto da lógica de predicados na qual OWL é parcialmente mapeada, com o objetivo de utilizar motores de inferência como o Jena [Jen12].

O trabalho de [Luc07] discute a rastreabilidade no processo de desenvolvimento de software utilizando técnicas de recuperação da informação para avaliar a similaridade entre produtos de trabalho. Eles apresentam como a Indexação Semântica Latente (LSI) pode ser utilizada para recuperar elos de rastreabilidade baseada em similaridade de texto presente nos artefatos de software. O trabalho também apresenta uma ferramenta que ajuda os engenheiros de software a descobrir elos de rastreabilidade emergentes durante a evolução do software, bem como monitorar elos previamente definidos. A proposta foi avaliada por um estudo de caso envolvendo 150 estudantes em 17 projetos. Os autores identificaram que a proposta recupera muitos falso positivos se a similaridade entre os artefatos vai além do intervalo mínimo definido. Por isso, a proposta é muito dependente de contexto, visto que este intervalo deve ser adaptado para cada projeto, ameaçando seu uso prático. Por outro lado, existe um efeito colateral de detectar possíveis artefatos mal documentados, visto que a ferramenta facilita a inspeção dos artefatos através de elos de rastreabilidade.

Em [Luc11], os autores apresentam uma abordagem que melhora o léxico do código fonte comparando anotações e comentários à terminologia da especificação de requisitos. Esta abordagem utiliza os benefícios de [Luc07] em aplicar técnicas de recuperação da informação para avaliar a similaridade entre requisitos e código fonte. A premissa geral neste artigo é que a similaridade entre artefatos de alto nível e o código fonte é um indicador da qualidade das anotações e comentários no código. A ideia é fornecer uma ferramenta para desenvolvedores (um *plugin* do Eclipse) que sugere termos do domínio que possam ser usados no código fonte durante as atividades de desenvolvimento. O artigo também apresenta uma avaliação empírica da ferramenta utilizando dois experimentos controlados, tendo como participantes alunos de mestrado e graduação. Pela sugestão de termos durante a codificação do software, a proposta tenta minimizar o impacto de termos não significantes introduzidos durante a implementação, os quais reduzem a identificação de similaridade na definição de elos de rastreabilidade.

O trabalho de Zhang et al. [Zha08a] utiliza técnicas de PLN e ontologias para mapear automaticamente código fonte e documentação. Os autores propõem uma ontologia de dois níveis: código fonte e documentação. A ontologia do código fonte contém conceitos tais como Classe, Variável e Comentários. A ontologia da

documentação é construída utilizando técnicas de PLN, incluindo algoritmos de reconhecimento de nomes de entidades e análise de estrutura sintática. Pelo elo entre estas ontologias, os autores são capazes de buscar no modelo relações entre o código fonte e documentação. A avaliação da proposta foi realizada utilizando um exemplo de uso e consultas neste exemplo. Os autores não apresentam nenhuma avaliação empírica da proposta nem como aplicar sua proposta em produtos de trabalho no nível de projeto. A documentação utilizada (JavaDoc) e o código fonte possuem um relacionamento bastante forte visto que ambos são gerados durante a fase de implementação. Seria interessante explorar o mapeamento do código fonte com os requisitos da aplicação utilizando ontologias geradas a partir do modelo de domínio.

Em Song et al. [Son08] propõe-se um método que combina LD (Lógica Descritiva) e Jena para desenvolver representações ontológicas uniformes para vários artefatos de software, desde o nível de requisitos, projeto e código fonte. Este componente analisa as relações correspondentes entre LD e OWL e desenvolve um conjunto de funções utilizadas para buscar informações na ontologia de manutenção do software. Os autores não apresentam como essa ontologia é desenvolvida nem mantida. Além disso, o envolvimento do papel do analista de domínio não é discutido conforme apresentado na Seção 3.1. Apesar dos autores sugerirem o mapeamento do código fonte com os requisitos, não é explorado como existentes técnicas poderiam ser utilizadas para extrair estas associações conforme explorado na Seção 3.3.2.

O trabalho de Assawamekin et al. [Ass09] também propõe utilizar ontologias para representar requisitos e especificações do software. Os autores exploram a perspectiva estrutural da arquitetura de software (usando diagramas de classe) para automaticamente gerar elos de rastreabilidade. A abordagem não cobre a criação e refinamento da ontologia, rastreabilidade de modelos de software comportamentais e a comparação léxica e semântica entre os termos. Não é apresentada também nenhuma avaliação empírica para verificar se a proposta gera elos de rastreabilidade falso positivos.

2.3. Recuperação de Informação e Análise de Impacto

A Recuperação da Informação lida com a representação, armazenamento, organização e acesso a informação. Diferente da recuperação de dados, que está preocupada em obter dados que satisfaçam determinado critério, a RI lida com a

obtenção de informação que nem sempre está estruturada e pode ser semanticamente ambígua [Bae99]. O campo de RI nasceu na década de 50, principalmente devido à crise em bibliotecas. Ao longo desses 50 anos, a área teve um avanço significativo devido aos inúmeros sistemas que utilizam RI diariamente para reduzir a sobrecarga de informação. Muitas universidades e bibliotecas públicas utilizam sistemas de RI para acessar livros, jornais e outros documentos. Mecanismos de busca como o Google [Goo12], Amazon [Ama12] e outros representam aplicações que se destacam na área.

Segundo Oliveto [Oli08], um processo de RI inicia quando um usuário submete uma consulta a um sistema. Consultas são declarações formais de uma necessidade como, por exemplo, busca de palavras em motores de busca. Em RI, uma consulta não identifica unicamente um objeto em uma coleção, mas vários objetos, tais como documentos texto, imagens ou vídeos, que podem satisfazer esta consulta com diferentes níveis de relevância.

Os primeiros sistemas de RI eram sistemas booleanos os quais permitiam recuperar informações utilizando combinações complexas de operações binárias. Atualmente, a maioria dos sistemas de RI utiliza a classificação para estimar a utilidade de um documento dada uma consulta, qualificando objetos quanto ao seu valor e relevância.

Segundo [Bae99], o processo associado a RI é composto por cinco estágios:

1. definição de uma base de dados e a visão lógica na qual informações serão recuperadas, incluindo documentos e suas estruturas;
2. indexação dessas estruturas;
3. especificação de uma consulta que reflita uma necessidade de um usuário;
4. busca dos documentos que satisfaçam a consulta; e, por fim,
5. definição de uma relação de relevância (*rank*) entre os documentos recuperados.

O último estágio corresponde à avaliação se os documentos retornados em uma consulta são aqueles de interesse, isto é, se o sistema conseguiu retornar os documentos que o usuário gostaria. Apesar de não ser um processo recente, este modelo é válido até os dias de hoje, mas com uma melhor performance computacional e efetividade [Mar09]. Em vias gerais, Marmanis e Babenko sugerem que para a captura de documentos é necessário realizar um *parser* para transformar os documentos alvo da busca em uma estrutura comum visando à indexação [Mar09].

2.3.1. Modelos de Recuperação de Informação

Baeza-Yates e Ribeiro-Neto apresentam em [Bae99] alguns modelos clássicos de RI como o booleano, vetorial e probabilístico conforme descrito a seguir.

O Modelo Booleano é um modelo de recuperação simples baseado na teoria de conjuntos e álgebra booleana. Este modelo apenas considera se índices de termos estão presentes ou não em documentos, o que implica que os pesos de cada índice se tornam binários, isto é, o documento é ou não relevante a busca. A pesquisa é realizada utilizando apenas por conectores lógicos como *and, or* e *not*. Este modelo sofre grandes limitações, tais como decisão binária que implica em recuperação de dados e não de informação, semelhante a bancos de dados convencionais. Além disso, este modelo carece de semântica, pois nem sempre documentos podem ser traduzidos como sim ou não.

O Modelo Vetorial reconhece que pesos binários são muito limitados e propõe um framework que torna possível mapear parcialmente consultas a documentos. Isso se torna viável pela atribuição de pesos não binários a índices de termos, computando assim um grau de *similaridade* entre consultas e documentos, recuperando informação de forma mais precisa que o Modelo Booleano. No Modelo Vetorial, pesos são atribuídos aos vetores consulta e documentos e sua correlação é definida através do ângulo entre os dois vetores através do grau de similaridade. Como vantagem, tem-se que:

1. o esquema de ponderação dos índices melhora a performance de recuperação;
2. a estratégia de mapeamento parcial permite a recuperação de documentos que se aproximam da consulta; e.
3. a fórmula pelo cosseno do ângulo ordena os resultados dos documentos retornados conforme o grau de similaridade da consulta.

Como desvantagem, os índices dos termos são assumidos como independentes uns dos outros, não considerando seu contexto semântico.

O modelo de Indexação Semântica Latente (*Latent Semantic Indexing* – LSI) é um mecanismo que induz a compreensão do significado de palavras pela análise das relações entre as palavras em grandes textos [Lam98]. Este modelo visa recuperar a ideia de um texto pelos conceitos descritos no mesmo ao invés de termos índices em sua descrição. O modelo LSI foi originalmente definido para resolver os problemas de polissemia e sinonímia que ocorre no Modelo Vetorial. O método utilizado para capturar a semântica essencial é a redução da dimensão, selecionando as dimensões mais

importantes a partir da matriz de coocorrência utilizando *Singular Value Decomposition* (SVD). SVD é uma forma de análise de fator que visa reduzir as dimensões de um espaço de uma funcionalidade sem perder especificidade. Uma das abordagens mais expressivas do SVD é o motor de busca do Google, na qual qualquer matriz pode ser decomposta para ser posteriormente recomposta utilizando a menor dimensão da matriz original.

O Modelo Probabilístico se baseia que dada uma consulta do usuário, existe um conjunto de documentos que contém exatamente os documentos relevantes e nenhum outro, considerados como o conjunto ideal. O processo de busca, neste contexto, corresponde a identificar exatamente as propriedades deste conjunto ideal. Como não é possível determinar essas propriedades durante a execução da consulta, o modelo sugere sua suposição. A partir desse momento, é necessária uma interação com o usuário para validar e sugerir melhorias desse conjunto inicialmente sugerido. Com base nessa avaliação do usuário, o sistema refina sua descrição do conjunto ideal. O Modelo Probabilístico é baseado fundamentalmente no princípio da pressuposição. Como desvantagem deste modelo, pode-se destacar:

1. a necessidade de uma pressuposição inicial para separa os documentos relevantes e não relevantes;
2. o método não leva em consideração a frequência que um termo ocorre no documento (os pesos são binários); e.
3. a adoção de pressuposições independentes para os índices de termos.

Sobre os modelos apresentados, o Modelo Booleano geralmente é considerado o modelo mais fraco. Seu maior problema é a incapacidade de reconhecer o mapeamento parcial, gerando assim uma performance inferior. Existem controvérsias entre o Modelo Vetorial e Probabilístico, pesquisadores como [Sal87] apresentam situações em que modelos possuem melhor desempenho em determinados problemas.

Um Modelo Probabilístico clássico é a Rede de Crenças Bayesiana, utilizada para evidenciar o relevância de um documento. Redes Bayesianas são grafos acíclicos direcionados no qual os nodos representam variáveis aleatórias e as arestas relações casuais entre essas variáveis. A força dessas influências casuais são expressadas através de probabilidades condicionais. Para expressar esta relação, considere x_i como um nodo de uma rede Bayesiana G e T_{xi} o conjunto de nodos pai de x_i. A influência de T_{xi} em x_i pode ser especificada por qualquer função $F_i(x_i, T_{xi})$ que satisfaça:

$$\sum_{\forall xi} Fi(xi, Txi) = 1$$

$$0 \leq Fi(xi, Txi) \leq 1$$

Onde x_i também se refere aos estados aleatórios da variável associada ao nodo x_i.

Redes de crenças fornecem os meios estatísticos para lidar com processos incertos, vagos e propensos a erro. A teoria fundamental por trás dessas redes é o teorema de Bayes, representado pela equação:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

Onde P(A,B) = P(B|A)P(A) é baseado na regra da cadeia. P(A) é a probabilidade inicial ou probabilidade marginal de um vetor A, o qual é independente do vetor B. P(A|B) é a probabilidade condicional de A, dado B.

As probabilidades condicionais de uma rede Bayesiana podem ser definidas através de diferentes estratégias. A estratégia de ordenação TFIDF, por exemplo, considera a normalização de frequência de termos (*TF, term-frequency*) e de documento inversa (IDF*, inverse-document-frequency*), cuja fórmula é:

$$w_{i,q} = \left(0.5 + \frac{0.5\, freq_{i,j}}{max_i\, freq_{l,q}}\right) \times \log\frac{N}{n_i}$$

Onde N é o número total de documentos do sistema, n_i é o número de documentos no qual o índice dos termos k_i aparece no documento d_i e $freq_{i,j}$ é a frequência bruta do termo k_i no texto da informação recuperada por q.

2.3.2. Algoritmos de Busca

A recuperação de informações através de consultas pode ser justificada pelo principio probabilístico de ordenação conforme modelo apresentado anteriormente. Os modelos clássicos pressupõe a ausência de conhecimento sobre os documentos [Zha08b]. Contudo, algoritmos modernos vêm explorando o conhecimento prévio de documentos incorporando preferências de ordenação estática, isto é, ordenação independente da consulta executada, tais como o HITS [Kle99] e PageRank [Bri98].

Dada uma consulta (um conjunto de palavras ou outros termos), HITS invoca um tradicional motor de busca para obter um conjunto de páginas relevantes, expandindo este conjunto utilizando *links* de entrada e saída e, então, tenta identificar dois tipos de páginas: *hubs*, páginas que apontam para várias páginas de alta qualidade, e *authorities*, que representam paginas de alta qualidade.

O algoritmo PageRank [Bri98], famoso algoritmo de busca do Google, foi introduzido em 1998 e representa um modelo de análise de elos. O algoritmo original melhora a ordenação dos resultados de uma consulta computando um simples vetor usando a estrutura de elos da Web para capturar a importância de cada página, independente de uma consulta particular [Hav03]. Esta independência é um grande avanço comparado ao HITS, cuja ordenação é calculada em tempo de consulta.

Medidas de RI tradicionais, como TFIDF, ordenam o documento baseado na frequência de termos. PageRank, no caso, ordena documentos baseado em sua popularidade, como quantos documentos apontam para um segundo documento e, intuitivamente, o mais popular possui uma maior relevância.

A ideia chave do PageRank é considerar hiperlinks de uma página para outra como recomendações ou popularidade. Então, quanto maior a popularidade de uma página, isto é, existem várias páginas a referenciando, maior sua importância.

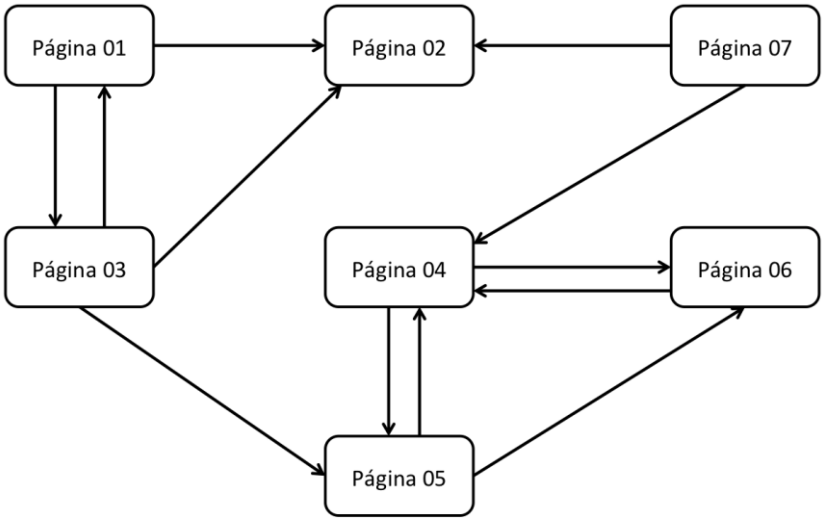


Figura 2.10 – Grafo direcionado representando elos entre páginas (adaptado de [Mar09]).

A Figura 2.10 adaptada de [Mar09] apresenta um grafo orientado para um domínio qualquer. A seta representa que a página de origem referencia a página de destino. Baseado nesta estrutura, é possível definir a *Matriz de Hiperlinks H* e um vetor *p*. Cada linha da matriz H é construída contando o número referências que uma página P_i

faz (digamos $N_{(i)}$) e atribuindo a coluna j o valor $^{1}/_{N_{(i)}}$ caso a página possua alguma referência, ou Zero caso contrário. Então para o grafo da Figura 2.11, tem a matriz H apresentada pela Figura 2.11.

0	1/2	1/2	0	0	0	0
0	0	0	0	0	0	0
1/3	1/3	0	0	1/3	0	0
0	0	0	0	1/2	1/2	0
0	0	0	1/2	0	1/2	0
0	0	0	1	0	0	0
0	1/2	0	1/2	0	0	0

Figura 2.11 – Matriz H (adaptada de [Mar09]).

Para calcular o PageRank, é utilizada a fórmula abaixo.

$$p(k+1) = p(k) * H$$

Sendo os valores *de p* o valor do PageRank de cada página do grafo e $p(0) = 1/n$. Por questões técnicas, a matriz H é geralmente substituída pelo Google pela matriz G, que possui melhores propriedades matemáticas. Na Web, existe um problema em que páginas não referenciam outras páginas, o que leva a problemas no algoritmo impedindo a navegação para outras páginas e correspondem as células que possuem valor Zero na matriz H. Para correção deste problema, o PageRank inclui um fator de ajuste que corresponde a um pulo aleatório, referenciado como um ajuste estocástico, atribuindo aleatoriamente as células com Zero o valor $1/n$, onde *n*, é o número de páginas no grafo. Outra restrição é que é possível alterar a navegação através de saltos arbitrários, chamados de *alpha*, que corresponde à primitiva de ajuste. O valor de *alpha* usado pelo Google é 0.85 [Mar09].

2.3.3. Processamento de Documentos

Conforme [Bae99], durante o processamento de documentos, nem todas as palavras são igualmente significantes para representação semântica de documentos, o que significa que na linguagem escrita, algumas palavras carregam mais significado que outras. Geralmente substantivos são as palavras mais representativas em um documento

devido a sua carga semântica. Então, é geralmente considerado útil o pré-processamento de texto de documentos em uma coleção visando determinar quais os termos candidatos a índices. O pré-processamento pode ser dividido em cinco principais operações: análise léxica, eliminação de *stopwords, stemming*, seleção de termos índices e construção das estruturas de categorização dos termos.

Análise léxica que corresponde basicamente em converter uma sequência de caracteres em uma sequência de palavras ou *tokens*. *Tokens*, neste caso, são grupos de caracteres com uma significância coletiva candidatos a termos índices [Fra92]. Em resumo, o maior objetivo da análise léxica é identificar palavras em um texto. Entretanto, existem palavras que, apesar de aparecerem inúmeras vezes no documento, não apresentam significado de relevância, como os artigos, as preposições, os pronomes e outras classes de palavras auxiliares. A esse conjunto de palavras não significantes é dado o nome de *stopwords* [Cro10].

Eliminação de *stopwords* que corresponde a filtrar palavras que possuem uma discriminação muito baixa no processo de recuperação [Bae99]. O objetivo deste processo é evitar que palavras insignificantes interfiram no processo de RI, reduzindo, assim, o tamanho do texto e do documento, o que facilita o armazenamento destes termos. A utilização de uma lista de *stopwords* bem elaborada permite que sejam eliminados muitos termos irrelevantes, aumentando, assim, a eficiência do resultado obtido pelo processo de indexação. Apesar de reduzir o texto, esta técnica pode apresentar problemas, pois nem sempre as palavras de maior ou menor frequência não são significativas para o contexto.

As palavras que compõem um documento podem ser variações de outras palavras utilizadas nas consultas (plural, grau, etc.). As diversas formas de flexionar palavras podem não alterar seu valor semântico e isso pode ser um problema no processo de busca. Para minimizar esse problema, foram desenvolvidas diversas técnicas que permitem aos buscadores identificar relações semânticas entre consultas e documentos. A conflação, o ato de fusão ou combinação, para igualar variantes morfológicas, é a técnica utilizada pelos buscadores que amplia essas relações semânticas.

De acordo com Croft, o stemming é o processo de reduzir a grande dimensionalidade de termos que são extraídos em um conjunto de documentos [Cro10]. Ele possui como objetivo remover os afixos (prefixos e sufixos) reduzindo o termo (*token*) a sua provável raiz (*stem*), possibilitando assim recuperar documentos equivalentes

apesar de pequenas variações sintáticas como, por exemplo, “conectando”, “conectado”, “conexão”, etc.

A seleção de termos índices visa determinar quais palavras/stems (ou grupos de palavras) podem ser utilizadas para indexar elementos. A decisão na escolha de uma palavra em particular para ser usada como termos índices está relacionada à natureza sintática da palavra e, de fato, substantivos são aquelas que frequentemente possuem uma maior semântica, se comparadas a adjetivos, advérbios e verbos [Bae99].

Por último, a construção das estruturas de categorização dos termos, tais como tesauros, permitem a expansão da consulta do termo para termos correlatos. Um tesauro provê um vocabulário preciso e controlado que serve para coordenar a indexação e recuperação de documentos, viabilizando assim a seleção dos termos mais apropriados [Sri92]. Basicamente, as vantagens correspondem à definição dos relacionamentos entre termos, identificação do número de ocorrência, especificação e normalização de um vocabulário comum.

2.3.4. Métricas para Recuperação de Informação

A definição de credibilidade de classificação de informação é geralmente o ponto inicial para avaliar os métodos empregados. Quando se submete uma consulta a um sistema de RI, o resultado pode não ser o esperado. Existem várias métricas para avaliar o grau de sucesso desses sistemas, sendo as mais comuns revocação e precisão [Fra92]. A Figura 2.12 adaptada de [Bae99] apresenta todas as possibilidades de resultados de uma consulta típica.

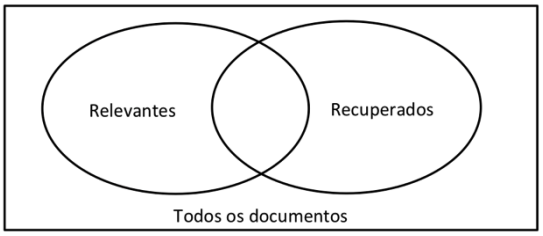


Figura 2.12 – Precisão e revocação para um exemplo de requisição de RI (adaptado de [Bae99]).

Conforme Figura 2.12, dado um conjunto com todos os documentos, um subconjunto desses documentos será relevante para a consulta do usuário e outro subconjunto será aquele recuperado por sua consulta. Claramente o objetivo é recuperar

todos os documentos relevantes, mas raramente este é o caso, portanto a intenção é recuperar a intersecção conforme ilustrado pela figura.

Revocação, por sua vez, mede a proporção de documentos relevantes recuperados, isto é, o número de documentos relevantes recuperados divididos pelo número total de documentos relevantes (Rt) conforme:

$$Revocação = \frac{RR}{Rt}$$

Quantitativamente, estas duas métricas respondem questões diferentes. A precisão responde a extensão em que se obtém aquilo que se gostaria e revocação responde se foi recuperado tudo aquilo que deveria ter sido. Percebe-se que é mais fácil encontrar a precisão que revocação, visto que para a segunda é necessário saber de antemão quais são todos os documentos relevantes dada uma consulta.

Uma medida que combina precisão e revocação é sua média harmônica [Bae99], também conhecida como medida F, e computada conforme:

$$F(j) = \frac{2}{\frac{1}{r(j)} + \frac{1}{P(j)}}$$

Onde r(j) é a revocação para o elemento j, P(j) é a precisão para o elemento j e F(j) é a média harmônica de r(j) e P(j).

2.3.5. Recuperação de Informação Aplicada a Análise de Impacto

Nos últimos anos, uma série de abordagens automatizadas para analisar impacto utilizando recuperação de informação vem sendo propostas. Helm et al. [Hel91] apresenta uma abordagem e uma ferramenta para construção automática de grandes bibliotecas de software a partir de documentação de seus componentes de software, fazendo uso do código fonte e documentação associada. A abordagem proposta combina dois tipos de análise: técnicas de RI baseadas em análise de documento e abordagens específicas de domínio, na qual a maioria da informação é fornecida por um especialista de domínio baseadas na análise de código. Nesta técnica, é realizado o *parser* do código fonte para derivar informação estrutural essencial como a relação entre entidades (por exemplo, classes, métodos e variáveis) – “derivadas de”, “membro de”, com base em um modelo de dados da linguagem de programação. Por outro lado, RI é utilizada para indexar documentação de software. Mais especificamente, o mecanismo de indexação da ferramenta desenvolvida é centralizado ao redor dos conceitos de relações léxicas e

noção de quantidade e qualidade da informação disponível ao especialista de domínio. A partir dessa abordagem, é possível a realização de consultas em linguagem natural para encontrar informação relevante no código fonte e documentação das aplicações analisadas.

Antoniol et al. [Ant00] propõe um método para avaliar um conjunto de componentes de sistema inicialmente impactados por uma mudança a partir de uma requisição textual. Duas abordagens foram aplicadas para recuperar os documentos relevantes a partir de uma requisição de mudança: o modelo vetor espacial e probabilístico. A abordagem foi estruturada em três passos:

1. o texto da requisição de mudança é utilizado para classificar os documentos associados com o ciclo de desenvolvimento e manutenção aplicando abordagens de RI;
2. os documentos disponibilizados são mapeados com áreas do código fonte; e
3. este mapeamento é utilizado para identificar componentes do código correspondentes aos documentos identificados pelo primeiro passo.

A proposta foi aplicada em classes de domínio público da biblioteca C++. Os autores chegaram à conclusão de que os maiores valores de revocação foram mais interessantes do que de precisão no que compete a qualidade dos resultados, visto que é importante não perder nenhum documento impactado por uma requisição de mudança.

A identificação de conceitos pode também ser útil para detecção de clones. Marcus e Maletic [Mar01] utilizam Indexação Semântica Latente (LSI – *Latent Semantic Indexing*) para analisar estaticamente o software e determinar similaridades semânticas entre documentos (funções, arquivos ou segmentos de código). Estas similaridades são usadas para o processo de detecção de clones de código fonte.

Em Chen et al. [Che01] é apresentada uma abordagem e ferramenta chamada CVSSearch que aproveita as características de ferramentas de gestão de configuração, na qual os comentários tipicamente descrevem as linhas de código que foram impactadas durante a persistência de um código fonte. Em particular, o CVSSearch permite que desenvolvedores pesquisem em comentários do CVS e fragmentos de código associados. A abordagem assume vantagem de que os comentários do CVS tipicamente descrevem as linhas de código envolvidas durante a operação de persistência e esta descrição é mantida para futuras revisões. Métodos de RI como *stemming* e classificadores são usados para armazenar e buscar as informações extraídas dos *logs* do CVS e linhas de código associadas na revisão dos arquivos através de análise de similaridade.

Canfora e Cerulo em [Can05] propõem um método que explora algoritmos de RI e um modelo probabilístico para associar uma descrição de requisição de mudança e um conjunto histórico de revisões de código fonte impactados por antigas requisições de mudanças similares. O raciocínio por traz da abordagem proposta é que a descrição textual mantida em uma ferramenta de gestão de erros (*bug tracking*) durante a resolução de uma requisição de mudança é um descritor útil dos arquivos impactados para requisições de mudanças futuras e similares. O artigo possui como hipótese de que um conjunto de comentários de revisão de um arquivo e um conjunto de requisições de mudanças que anteriormente impactaram esse arquivo são bons descritores dos arquivos que apoiam a análise de impacto de novas requisições de mudanças. O artigo descreve duas contribuições:

1. a informação histórica armazenada em uma ferramenta de gestão de erros e a informação histórica armazenada em uma ferramenta para gestão de configuração, como CVS, são úteis para analisar os aspectos relacionados à evolução do software bem como sua análise de impacto; e
2. a informação armazenada em requisições de mudanças históricas são descritores úteis do código fonte que podem ser utilizados para a análise de impacto através de análise textual de similaridade.

Uma das tarefas mais importantes da análise de impacto é a localização de conceitos. Esta localização é geralmente realizada por um processo intuitivo e informal cujo objetivo é identificar partes do software que implementam certa funcionalidade ou conjunto de conceitos. Esta atividade compreende um pré-requisito para diversas tarefas de evolução e compreensão do programa, sendo ela uma atividade desempenhada comumente por desenvolvedores (identificar como conceitos e entidades do domínio são implementadas). A atividade de localização de conceitos é uma das aplicações mais comuns dos métodos de RI no contexto da Engenharia de Software [Oli08].

A localização de conceitos no código fonte utilizando LSI e PageRank é abordado por Marcus et al [Mar04a, Mar04b]. Em particular, o usuário formula uma consulta, que consiste em uma série de palavras que descreve um conceito a ser localizado. A técnica usa esta consulta para classificar e recuperar os documentos de um sistema, como arquivos, classes, métodos ou funções. Por exemplo, para a frase “Adicionar pagamento por cartão de crédito no sistema de vendas”, a técnica identificaria os conceitos “pagamento” e “cartão de crédito” no código fonte do programa. Em [Pos06a] é apresentada uma ferramenta para localização de conceitos que utiliza a ferramenta *off the*

shelf Google Desktop Search. A abordagem é implementada como um *plugin* do Eclipse e chamada *Google Eclipse Search* (GES). GES permite aos desenvolvedores pesquisar nos projetos de software de forma similar a uma busca na internet. Esta abordagem visa resolver problemas utilizando para tanto ferramentas e frameworks existentes (como a API - A*pplication Programming Interface* - do Google Desktop Search) para busca de conceitos.

JSearch [Sin06] é uma ferramenta projetada para localizar partes comum de código fonte em um repositório utilizando RI para indexar informações extraídas do código fonte. JSearch não indexa os arquivos como documentos planos, ele realiza o *parser* do código fonte utilizando um Parser Java AST e extrai os campos de cada classe para indexação, fazendo uso da API Java Lucene. A ferramenta não processa linguagem natural para realizar a consulta, mas possui uma linguagem específica de domínio na qual os desenvolvedores podem especificar os membros do código, tais como classes, nomes de métodos, nos quais a busca será feita.

Uma abordagem estática que não necessita de interação com o usuário, chamada SNIAFL, é apresentada em Zhao et al. [Zha06]. Esta abordagem utiliza RI e análise estática para automaticamente associar descrições textuais de funcionalidades com sua localização no código fonte. SNIAFL usa RI para revelar conexões básicas entre das funcionalidades e unidades computacionais (como funções) no código fonte. Como a recuperação inicial pode ser imprecisa, SNIAFL usa a representação do grafo de chamada para restringir funções que não são relevantes identificadas inicialmente. A novidade da abordagem é o uso combinado dos tipos de análise para endereçar a identificação de funcionalidades: RI e análise estática.

A performance da localização de conceitos pode ser melhorada combinando diferentes técnicas. Poshyvanyk et al. [Pos06b] combinam duas técnicas para localização de conceitos: classificação probabilística baseada em cenário (*scenario-based probabilistic ranking* - SPR) e LSI do código fonte. Todos os métodos e funções são extraídas do código fonte (incluindo os comentários) e indexados com LSI. A combinação apresentada dos métodos aumentou significativamente a precisão na identificação do ponto inicial de implementação de uma funcionalidade durante o processo de localização. O estudo de caso apresentado demonstrou que a abordagem combinada teve um melhor desempenho que o uso das mesmas técnicas isoladas.

Poshyvanyk e Marcus também apresentam uma abordagem combinando LSI e *Formal Concept Analysis* (FCA) em conjunto [Pos07]. FCA é utilizada para definir o

conceito como uma unidade de duas partes: extensão e intenção. A extensão de um conceito recupera todos os objetos que pertencem ao conceito, enquanto a intenção referencia todos os atributos que são compartilhados pelos objetos em consideração. Na abordagem proposta, LSI é utilizada para mapear os conceitos recuperados em consultas escritas pelo programador com as partes do código fonte, apresentando uma lista ordenada dos resultados. Dada esta lista ordenada, a abordagem seleciona os atributos mais relevantes destes documentos e organiza os resultados via FCA. Em particular, a proposta é utilizada para mapear conceitos expressos por consultas escritas pelos programadores para as partes de código relevantes, apresentando ao fim uma lista ordenada de resultados de busca. Dada esta lista ordenada, a abordagem seleciona os atributos mais relevantes destes documentos, organizando os resultados da busca em tópicos (categorias).

Duas técnicas de localização de conceitos são combinadas em Shepherd et al. [She07]: uma abordagem leve de PLN e análise de programa. O usuário interage com a ferramenta da seguinte forma: formulação da consulta inicial, a ferramenta expande esta consulta capturando ações relacionadas entre os identificadores do programa (considerando a relação verbo-ação), resultando em grafo orientado no qual não apenas os conceitos são identificados, mas as ações associadas a estes conceitos. A consulta é realizada sobre este grafo para descobrir relações estruturais entre os métodos utilizando como critério inicial os conceitos presentes na consulta. O diferencial da proposta é que utiliza técnicas de PLN para buscar a morfologia, sinônimos e estrutura das sentenças.

2.4. Revisão Sistemática

Com o objetivo de identificar o estado da arte em análise de impacto e sua integração com ontologias e recuperação de informação, foi desenvolvida uma revisão sistemática. A motivação para sua realização surgiu de estudos preliminares que não identificaram uma proposta que, a partir de uma perspectiva de negócio, identificassem informações relacionadas a estruturas de código fonte impactadas por determinada requisição de mudança. Surgiu então o interesse de relacionar ontologias, que capturam essa perspectiva de negócios, recuperação de informação, que obtém informações que nem sempre estão estruturadas e que podem ser semanticamente ambíguas, com o código fonte de uma aplicação. Assim, definiu-se a seguinte questão de pesquisa para

nortear a revisão: “*Quais são os principais resultados encontrados em pesquisas sobre análise de impacto utilizando ontologias e recuperação de informação?*”.

Em se tratando de análise de impacto, as Seções 2.2.4 e 2.3.5 apresentaram respectivamente um conjunto extensivo de trabalhos relacionados a ontologias e recuperação de informação. Ao responder a questão definida por esta revisão sistemática, estará se verificando o estado da arte sobre propostas que representam a intersecção dessas duas áreas e se há contribuição suficiente para caracterizá-la como uma área de pesquisa.

Apesar da abrangência do tema de análise de impacto, ontologias e recuperação de informação, a população dessa revisão se restringiu à intersecção desses conceitos conforme definição da questão de pesquisa. Para representar esta população, a Tabela 2.1 apresenta os principais termos considerados e alguns sinônimos.

Tabela 2.1 – Palavras chave e sinônimos.

Palavra chave (em inglês)	**Tradução em português**	**Sinônimos (em inglês)**
impact analysis	análise de impacto	*traceability, dependency*
ontology	*ontologia*	*domain model*
information retrieval	recuperação de informação	-

Adicionalmente às palavras chave, considerou-se como critério de intervenção o contexto de desenvolvimento de software e código fonte. Após a realização de alguns testes comparativos para avaliar a relevância dos resultados, foram utilizadas as expressões descritas na Tabela 2.2.

Tabela 2.2 – Expressões de busca.

Contexto	***String* de busca**
Análise de impacto, ontologia e recuperação de informação	(("impact analysis") OR ("traceability") OR ("dependency")) AND((ontology) OR (ontologies) OR ("domain model")) AND ("information retrieval") AND (("source code") OR ("software development"))

Simultaneamente ao processo de elaboração dos critérios de busca, foram selecionadas as seguintes os seguintes domínios para pesquisa:

1. ACM: http://portal.acm.org
2. IEEE: http://ieeexplore.ieee.org
3. Scopus: http://www.scopus.com
4. ScienceDirect: http://www.sciencedirect.com

O número de artigos retornados em Março e Abril de 2012 em cada fonte é apresentado na Tabela 2.3.

Tabela 2.3 - Número de artigos em cada fonte para expressão de busca.

ACM	IEEE	Scopus	Science Direct	Total
31	21	14	25	87 (eliminando repetições)

Para definirmos um critério de inclusão e exclusão dos trabalhos a serem selecionados, foi feita uma classificação com três níveis para enquadrar os trabalhos que atenderam a string de busca nos mecanismos de pesquisa definidos. Os níveis definidos para inclusão e exclusão dos artigos na revisão foram elaborados para serem aplicados em todos os artigos que foram retornados nos mecanismos de busca selecionados, como apresentado na Tabela 2.4.

Tabela 2.4 – Classificação dos artigos.

Classificação	Descrição
C1	O foco do artigo é análise de impacto utilizando ontologias e técnicas de recuperação de informação.
C2	O foco do artigo é análise de impacto utilizando ou ontologias ou técnicas de recuperação de informação.
C3	O foco do artigo é análise de impacto não utilizando ontologias nem técnicas de recuperação de informação.
C4	O foco do artigo não é análise de impacto.

Para a seleção dos trabalhos, foi definido um processo que iniciou com a aplicação da string definida nos mecanismos de busca/indexação escolhidos e finalizou com a escolha dos artigos de onde foram retiradas as sínteses e as contribuições para a revisão sistemática. Não foi feita nenhuma análise prática das aplicações ou protótipos descritos nos trabalhos selecionados. Também não foi incluída a identificação de ferramentas durante a fundamentação deste estudo. A Tabela 2.5 mostra o processo de seleção dos estudos e o número de artigos identificados em cada estágio.

Tabela 2.5 – Processo de seleção dos estudos.

Estágio	Tarefa	Resultado
1	Aplicação da string aos mecanismos de busca selecionados, considerando titulo, palavras-chave e resumo.	87 artigos encontrados
2	Os 87 artigos foram importados para ferramenta Mendeley [Men12] e classificados por um dos critérios (C1, C2, C3 ou C4 de acordo com a Tabela 2.4) pela leitura do titulo, *abstract* e	Artigos classificados: - C1: 05 - C2: 11 - C3: 14

	palavras-chave.	- C4: 57
3	Exclusão dos artigos classificados como 3 e 4.	16 artigos restantes.
4	Leitura da Introdução e Conclusão dos artigos classificados como 1 ou 2 e exclusão dos artigos que não contribuem para o foco da pesquisa.	11 artigos restantes.
5	Leitura dos 11 artigos restantes e avaliação de qualidade.	5 artigos eliminados pela avaliação de qualidade. 6 artigos restantes usados como estudos primários para a síntese dos estudos encontrados.

O resultado do processo de seleção é descrito pela Tabela 2.6. O Apêndice 1 descreve todos os artigos encontrados, categorizados conforme a Tabela 2.4, descrevendo inclusive o estágio/critério no qual cada trabalho foi ignorado conforme Tabela 2.5.

Tabela 2.6 – Artigos selecionados pela revisão sistemática.

Ref.	Classif.	Referência
[Ary11]	C1	Aryani, A., Perin, F., Lungu, M., Mahmood, A. N., & Nierstrasz, O. (2011). **Can we predict dependencies using domain information ?** 18th Working Conference on Reverse Engineering. doi:10.1109/WCRE.2011.17
[Nar11]	C1	Narayan, N., Bruegge, B., Delater, A., & Paech, B. (2011). **Enhanced traceability in model-based CASE tools using ontologies and information retrieval**. 2011 4th International Workshop on Managing Requirements Knowledge (pp. 24-28). Ieee. doi:10.1109/MARK.2011.6046559
[Ril06]	C1	Rilling, J., Zhang, Y., Meng, W. J., & Witte, R. (n.d.). **A Unified Ontology-Based Process Model for Software Maintenance and Comprehension**. Viewpoints (pp. 1-10).
[Bor11]	C2	Borg, M. (2011). **Do Better IR Tools Improve the Accuracy of Engineers ' Traceability Recovery ?** MALETS '11 (pp. 27-34).
[Kuh07]	C2	Kuhn, A., & Gı, T. (2007). **Semantic clustering : Identifying topics in source code**. Information and Software Technology, 49, 230-243. doi:10.1016/j.infsof.2006.10.017
[Tan11]	C2	Tang, A., & Vliet, H. V. (2011). **Software Architecture Documentation : The Road Ahead**. IEEE/IFIP Conference on Software Architecture (pp. 252-255). doi:10.1109/WICSA.2011.40

A síntese dos resultados representa um resumo dos principais conceitos encontrados nos trabalhos selecionados lidos integralmente por fazerem parte do foco do trabalho. A síntese dos estudos desta revisão sistemática foi representada pela Tabela 2.7.

Tabela 2.7 – Síntese dos resultados da revisão sistemática.

	Técnica		**Avaliação**
Ref.	**Recuperação Informação**	**Ontologia**	**Experimento**
[Ary11]	TFIDF	Conceitos	Medida F, Coef. Jaccard
[Nar11]	Frequência	Ontologia de Aplicação	–
[Ril06]	–	Ontologia de Aplicação	–
[Bor11]	Vetor Espacial, TFIDF,	–	Medida F, TOST

	Stemming		
[Kuh07]	LSI, TFIDF	WordNet	Medida F
[Tan11]	–	Ontologia de Aplicação	-

Em [Ary11] é apresentado uma abordagem para predizer dependências de software baseado no acoplamento de informações de domínio. Para tanto, são definidos *clusters* conceituais pela identificação de variáveis e funções de domínio no código e componentes de interface com usuário. A definição destes *clusters* corresponde à ocorrência de conceitos na camada da aplicação que faz a interface com o usuário, identificando o acoplamento de variáveis do domínio. Este acoplamento é representado com um grafo, o qual é ponderado usando o coeficiente de Jaccard para calcular o peso da ocorrência do termo em um contexto. Esta proposta utiliza a frequência de termos de conceitos do domínio para a análise de dependência, que inclui a identificação de classes, atributos e métodos navegando pelo grafo de conceitos até o código fonte. Esta abordagem foi avaliada utilizando uma conhecida ferrramenta ERP (*Enterprise Resource Planning*) de código aberto para predizer dependências, porém não foi apresentada nenhuma avaliação empírica comparando os resultados da proposta com resultados onde desenvolvedores predizem os impactos, baseando em sua experiência.

A proposta apresentada em [Nar11] cria e utiliza uma ontologia específica do projeto derivada a partir da informação textual de projeto com o objetivo de apoiar o processo de criação de elos de rastreabilidade. Para tanto, os autores propõem uma arquitetura composta por um Analisador de Artefato, que extrai informação de instâncias de artefatos para identificar subordinação de entidades, como atributos de classes. O módulo de Extração de Entidade Global utiliza um *tagger* para identificar substantivos de estruturas. Estes substantivos são utilizados pelo Construtor de Artefato Ontologia e Construtor Genérico de Ontologia para criação da ontologia a partir dos módulos anteriores, bem como a rastreabilidade entre os elementos. O módulo de Raciocínio Ontológico de Projeto combina as ontologias geradas para recuperar estruturas de código relevantes considerando frequência e importância. Não é apresentada nenhuma avaliação empírica.

Em [Ril06], os autores apresentam um modelo de processo formal para apoiar a manutenção de software. O modelo fornece uma representação ontológica que suporta o uso de serviços de raciocínio entre diferentes fontes de conhecimento. O modelo é composto utilizando o Racer e sistemas de mineração de texto entre a ontologia do código fonte e da documentação. Não foi apresentado nenhuma avaliação empírica.

Os autores de [Bor11] conduziram experimentos de abordagens existentes de rastreabilidade que fazem uso de recuperação de informação para avaliar a qualidade e acurácia das propostas. O resultado do experimento foi inconclusivo devido ao pequeno número de participantes, não permitindo rejeitar a hipótese nula de que a acurácia da rastreabilidade das ferramentas diferem. Os autores sugerem que é necessário maior investimento para obter retorno de investimento significativo pelo uso de técnicas de RI para rastrear objetos.

Em [Kuh07], os autores propõem o uso de recuperação de informação para explorar a informação linguística encontrada no código fonte tais como identificadores e comentários. Os autores propõem clusters semânticos para agrupar artefatos de software que utilizam um vocabulário similar, técnica essa baseada em LSI. Os elos de rastreabilidade são definidos pelos clusters semânticos, classificados pela frequência dos termos. Ao final são apresentados alguns estudos de caso que evidenciaram que a abordagem proposta fornece um bom apoio para desenvolvedores não familiarizados com o sistema de encontrar estruturas de código relevantes.

Os autores em [Tan11] propõem o uso de ontologias para melhorar a recuperação e rastreabilidade do conhecimento sobre o software fazendo uso de anotação semântica em código. O processo de indexação inclui a construção de uma ontologia preliminar gerada automaticamente a partir de requisitos funcionais e não funcionais, identificando instâncias de conceitos através de substantivos. A recuperação das estruturas impactadas se dá por uma descrição de cenários, os quais conceitos são identificados. Não foram apresentados resultados empíricos do uso da proposta.

Analisando todas as propostas resultado da revisão sistemática, não foi possível identificar trabalhos que utilizam efetivamente os recursos fornecidos por linguagens de representação do conhecimento, modelos de recuperação de informação e que tenham uma avaliação empírica relevante e consistente.

2.5. Considerações sobre o Capítulo

As seções anteriores apresentaram uma fundamentação teórica buscando contextualizar as atividades relacionadas à análise de impacto no contexto do desenvolvimento de software orientado a ontologias. Inicialmente foram apresentadas algumas definições sobre o termo análise de impacto bem como a relevância da área.

A seguir, foram apresentados alguns métodos para a execução da análise de impacto e suas principais etapas. Com relação às etapas, foram descritas as análises de rastreabilidade e dependência. A primeira define o relacionamento entre termos, enquanto a segunda avalia a relevância desses termos frente a uma solicitação de mudança. Com relação às etapas descritas, a primeira apresentou um estudo sobre as perspectivas, classificações, sintaxe e semântica dos elos de rastreabilidade. A análise de dependência, por sua vez, apresentou a sistemática das três principais técnicas: grafos de chamadas, particionamento estático e particionamento dinâmico.

A análise de impacto é consequência da evolução do software, direcionada por propostas de mudanças. Assim, foram apresentados os conceitos e procedimentos relacionados à gestão de mudanças durante o desenvolvimento de software. Mudanças são geralmente resultado de necessidades de negócio do mundo real, isto é, onde o software é posto em operação. Para modelar conceitos do mundo real, ontologias emergem como o principal formalismo para especificação explicita de conceitos. Foi então apresentada uma definição comum para o termo bem como a estrutura da linguagem OWL para sua representação.

Em Ciência da Computação, apesar de ontologias terem origem na área de Inteligência Artificial, este modelo é bastante difundido na área de Engenharia de Software. Foram então discutidos trabalhos relacionados a ontologias no contexto de análise de impacto e rastreabilidade.

Para contextualizar o escopo dessa proposta, foi apresentado o trabalho desenvolvido em [Nol07a] a respeito de rastreabilidade ontológica sobre o Processo Unificado. Observou-se que os elos de rastreabilidade interferem na análise de dependência e, consequentemente, são determinantes para uma adequada análise de impacto.

Este trabalho tem como hipótese que ao aprimorarmos os elos de rastreabilidade utilizando ontologias, a análise de impacto também poderá ser aprimorada. Sugere-se utilizar como base elos de rastreabilidade apoiados por conceitos com semântica associada, ao invés de simples e estáticos, como normalmente descritos nas propostas de análise de impacto, para melhorar assim a exatidão e cobertura na recuperação de alterações no código fonte frente a uma requisição de mudança.

O uso de ontologias para esse fim se mostra interessante, pois cada entidade computacional é derivada, direta ou indiretamente, de conceitos do mundo real.

Relacionando estes conceitos a estruturas de código fonte, permite-se uma análise dinâmica, pela dependência semântica entre os termos, bem como estática, pela organização estrutural do código. Para analisar estas dependências, foram apresentados conceitos sobre RI para busca de informação não estruturada no código fonte da aplicação. Foram apresentados modelos e algoritmos, técnicas de processamento de documentos e linguagem natural, métricas para avaliar a eficiência de modelos de RI. Também foi apresentada uma extensa relação de trabalhos que associam análise de impacto e recuperação de informação.

Este capítulo também apresentou uma revisão sistemática que visou identificar propostas de análise de impacto que integrem ontologias e modelos de recuperação de informação. Os resultados não foram satisfatórios ao responder a questão de pesquisa definida, evidenciando que a intersecção entre as áreas apresenta lacunas, conforme sumarizado pela Tabela 2.7. Este resultado encoraja o desenvolvimento de pesquisa na área.

Esta tese tem como base algumas evidências empíricas positivas quanto à definição e recuperação de elos de rastreabilidade entre conceitos da ontologia e elementos de diagramas da UML [Nol09]. Estas evidências instigam um estudo mais específico de quão relevante é a lista de termos associados frente uma necessidade de negócio, bem como a avaliação desta dependência em nível de código fonte conforme descrito no próximo capítulo.

3. MODELO DE ANÁLISE DE IMPACTO EM CÓDIGO FONTE UTILIZANDO ONTOLOGIAS

Este trabalho apresenta um modelo de análise de impacto em código fonte fazendo uso de ontologias. O objetivo é automatizar a rastreabilidade entre estruturas de código com o domínio da aplicação e identificar a relevância dessas associações frente a um requisito de mudança, considerando as categorias de manutenções definidas por [Cha00].

A Figura 3.1 ilustra o Modelo de Análise de Impacto proposto. Como entrada deste modelo, deve-se informar o código fonte da aplicação, uma ontologia que descreve seu domínio e um requisito de mudança. Apesar do desenvolvimento da ontologia de domínio fugir do escopo desta tese, uma proposta de processo de integração de ontologias com a engenharia de software é discutida no Apêndice C. A saída irá evidenciar as estruturas de código (classes, métodos ou atributos) que potencialmente serão impactadas. Este modelo é composto por dois submodelos: Modelo de Rastreabilidade e Modelo Probabilístico.

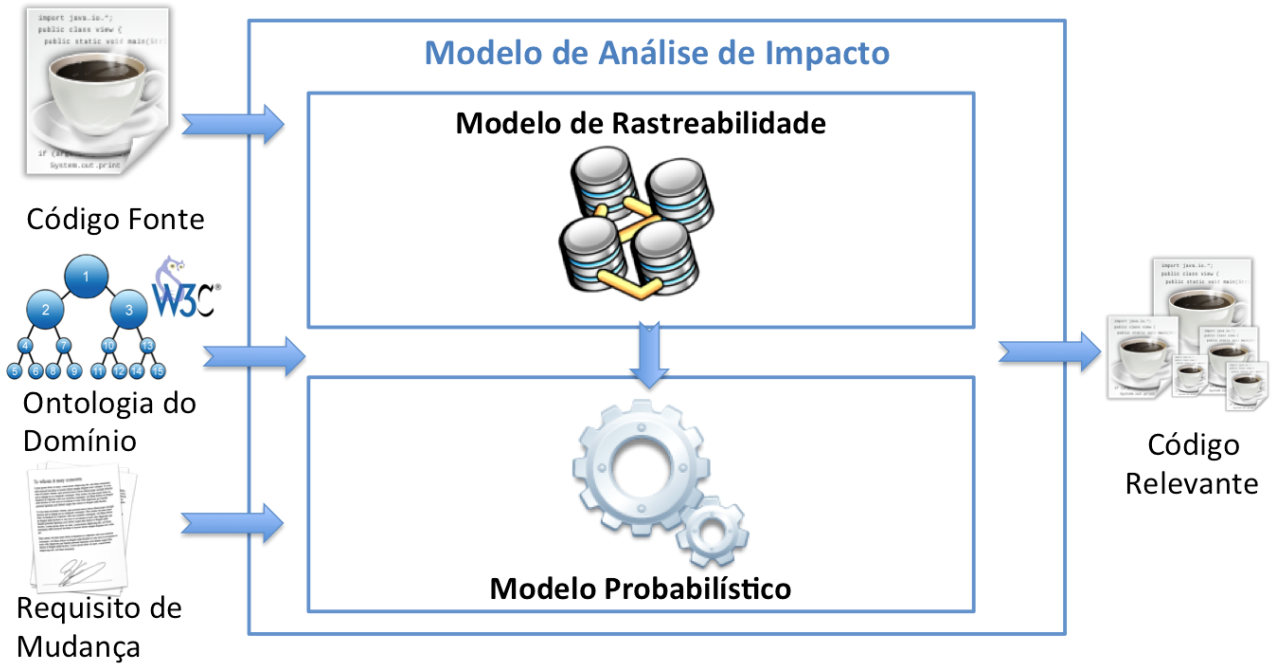


Figura 3.1 – Modelo de Análise de Impacto.

O Modelo de Rastreabilidade é responsável por interpretar a ontologia do domínio e o código fonte da aplicação. A árvore de sintaxe abstrata do código fonte é traduzida em uma nova ontologia, que possui seus componentes rastreados com os conceitos do domínio. Os elos de rastreabilidade são populados automaticamente na ontologia de

domínio através de um analisador de similaridade léxico e semântico, relacionando estruturas do código fonte (classes, métodos e atributos) às suas respectivas estruturas conceituais. Esta análise de similaridade inclui eliminação de *stopwords*, a categorização de cada termo do código fonte como substantivo, normalização conceitual (WordNet e glossário) e comparação utilizando Stemming. Este modelo recebe o código fonte e a ontologia do domínio como entrada e popula esta ontologia com elos de rastreabilidade.

O Modelo de Probabilidade avalia a relevância de cada elo de rastreabilidade utilizando o modelo probabilístico de recuperação de informação de Redes de Crenças Bayesianas. Este modelo utiliza diferentes técnicas para identificar a relevância de código, desde o algoritmos de classificação como o PageRank e TFIDF, até a análise de dependência conceitual utilizando grafos de chamada. Esta análise considera toda a estrutura de baixo nível do código fonte, respeitando sua organização léxica e semântica, sugerindo ao final a probabilidade de impacto das classes ou membros identificados como relevantes.

O Modelo de Análise de Impacto é relevante em diferentes aspectos. Primeiro, ele relaciona conceitos do domínio com o código fonte da aplicação através de uma rastreabilidade orientada a ontologias que viabiliza a recuperação de relacionamentos implícitos e explícitos utilizando motores de inferência.

Segundo, ele provê uma abordagem de população de ontologias automática usando processamento de linguagem natural para identificar instâncias de conceitos do domínio a partir de produtos de trabalho de software. Esta abordagem cria elos de rastreabilidade entre conceitos do domínio e o código fonte da aplicação.

Terceiro, ele pondera todos os elos de rastreabilidade utilizando um modelo probabilístico de Redes de Crenças Bayesianas. Além disso, ele avalia diferentes aspectos, desde a frequência que termos do domínio são referenciados em uma classe específica do código fonte, a frequência que um conceito específico aparece em todas as classes (frequência inversa do documento) e se a distância entre conceitos do domínio é equivalente a distância entre as classes do código referenciadas por estes conceitos (dependência conceitual).

Este capítulo também apresenta um cenário motivacional que descreve o problema relacionado à análise de impacto e que será utilizado para ilustrar cada aspecto do modelo proposto.

3.1. Cenário Motivacional

Para ilustrar o contexto deste trabalho, esta pesquisa utilizou um sistema descrito e implementado em [Arr01], que fornece descrições de casos de uso, diagramas UML e código fonte da aplicação. Trata-se de um sistema de apropriação de horas de funcionários chamado Sistema de Cartão Ponto (*Timecard*). O objetivo do sistema é apoiar o registro de horas em determinadas atividades, chamadas de Códigos de Débito (*charge code*). Neste sistema, os usuários podem realizar a manutenção desses códigos, a manutenção de empregados, bem como o registro de horas nesses códigos. Para ilustrar melhor o universo de discurso, a Figura 3.2 apresenta o Modelo de Domínio com os principais conceitos da aplicação e seus relacionamentos.

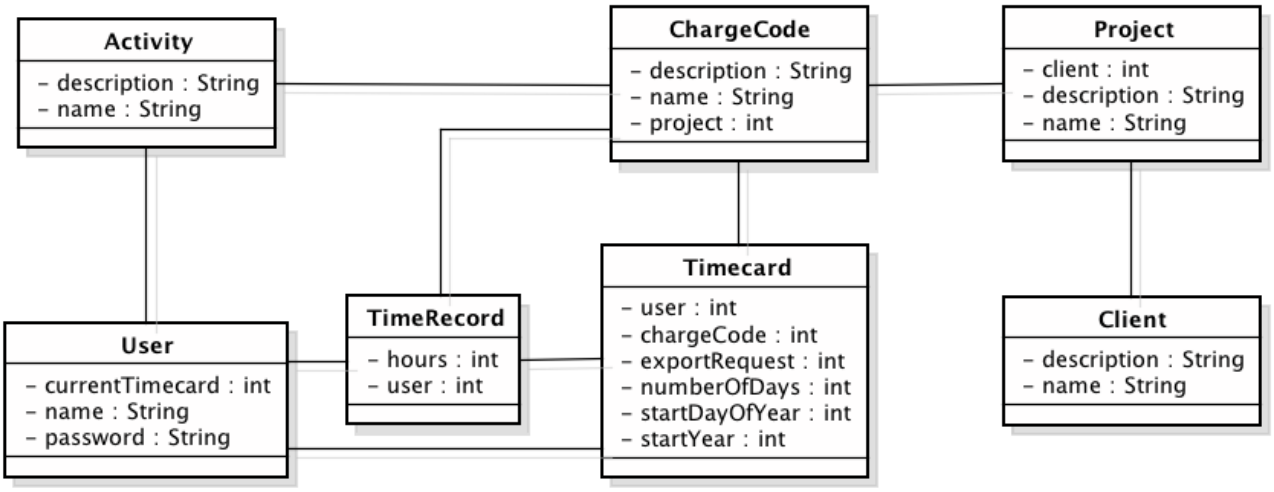


Figura 3.2 – Modelo de Domínio do Sistema de Cartão Ponto [Arr01].

Para compreender as principais funcionalidades oferecidas pelo sistema, o diagrama de casos de uso é apresentado na Figura 3.3. O exemplo seguido neste trabalho se refere ao caso de uso *Registrar Horas*, cujo fluxo principal se encontra descrito na Figura 3.4.

Durante a evolução do software, devido a uma necessidade de negócio, surge um requisito de mudança formalizado em um RdM. Este requisito caracterizado como mantenção evolutiva consiste em alterar o sistema a partir da declaração representada pela Figura 3.5.

Para entender um pouco o desafio em avaliar o impacto dessa mudança no código fonte, é necessário considerar aspectos técnicos bem como a complexidade do sistema. Com relação à tecnologia utilizada, o sistema foi desenvolvido em Java utilizando

Enterprise Java Beans (EJB), versão 1.1, o qual faz parte da especificação Java 2 Enterprise Edition (J2EE). O código é estruturado pelos seguintes componentes: *Entity bean, Home interface, Session bean, Remote interface, Implementation, Deployment descriptor, Bean-managed persistence* e *Container-managed persistence*.

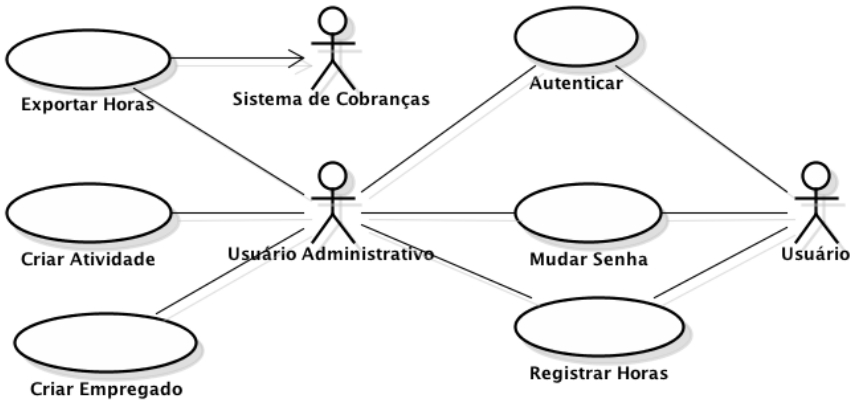


Figura 3.3 – Diagrama de Casos de Uso do Sistema de Cartão Ponto traduzido [Arr01].

Fluxo de eventos principal: Empregado registra suas horas.	
Passo #	**Ação**
1	O empregado visualiza as horas previamente entradas para o período atual.
2	O empregado seleciona um código de débito dentre os disponíveis, organizados por clientes e projeto.
3	O empregado seleciona um dia na semana corrente.
4	O empregado informa horas trabalhadas como um número decimal positivo.
5	As horas são armazenadas e visualizadas em qualquer acesso subsequente.

Figura 3.4 – Fluxo principal do caso de uso Registrar Horas [Arr01].

Durante o registro de horas, o usuário poderá informar horas para o mês corrente até a data atual.

Figura 3.5 – Requisito de mudança.

Com o objetivo de avaliar a complexidade do código no momento de analisar o impacto da mudança, foram extraídas algumas métricas apresentadas nas Tabelas 3.1 e 3.2. Estas métricas se tornam relevantes para analisar o escopo total da aplicação bem como a quantidade de código fonte passível de inspeção.

Tabela 3.1 – Estrutura do sistema.

Quantidade de Pacotes	Quantidade de Classes	Quantidade de Associações	Quantidade de Generalizações
12	75	13	21

Tabela 3.2 – Análise de Linhas de Código (LoC) do Sistema de Cartão Ponto.

Linguagem	Arquivos	Linhas em Branco	Linhas de Comentários	Linhas de Código
Java	75	890	721	3414
HTML	5	37	28	117
XML	1	0	0	12
DOS Batch	2	0	0	5

Pela análise das Tabelas 3.1 e 3.2, identificam-se 75 classes organizadas em 12 pacotes. Para realizar a análise de impacto da mudança solicitada, a Seção 2.1.3 apresenta algumas estratégias.

A primeira opção é a força bruta, navegando no programa, abrindo e fechando arquivos relacionados. Os desenvolvedores geralmente alteram o código para avaliar o impacto na aplicação durante sua execução. Esta opção é uma atividade manual fortemente baseada na experiência. No pior caso, o desenvolvedor deveria considerar as 3414 linhas de código, incorrendo em um exercício de tentativa e erro.

Uma alternativa à navegação em um programa é apoiar a análise na especificação do software. Para tanto, é necessário que a documentação do software esteja atualizada. De posse destes documentos, pode-se navegar por todos os modelos produzidos para analisar o impacto. Considerando a UML como referência, esta atividade inclui investigar (1) Diagramas Estruturais, como Diagrama de Classes, Objetos, Componentes, Infraestrutura, Pacotes, etc., e (2) Diagramas Comportamentais, como Diagramas de Atividades, Casos de Uso, Máquina de Estados, Sequência, Comunicação, etc. Uma vez identificado os impactos nos modelos, cada elemento deve estar mapeado adequadamente com as estruturas de código fonte para, então, realizar a análise de impacto.

O uso de matrizes de rastreabilidade convencionais pode apoiar a análise de impacto, relacionando requisitos com estruturas de código. Devido à granularidade do caso de uso, é possível identificar que um conjunto de classes esteja relacionado a um requisito. Não se pode, porém, considerar aspectos mais específicos do comportamento como, no exemplo em questão, o impacto específico em métodos e atributos. Além dessa restrição, deve-se avaliar se os relacionamentos entre requisitos e classes do sistema estão completos e corretos.

Por último, a análise de dependência é uma alternativa viável quando se refere a código fonte. Esta análise pressupõe navegar pelos métodos através do grafo de

chamadas. Para tanto, é necessário conhecer os métodos e considerar o impacto um a um. Com o objetivo de melhor entender a dinâmica da análise de dependência e particionamentos, a seguir será apresentada a ferramenta JRipples[2] descrita em [Buc05, Pet09]. Esta ferramenta é utilizada para a análise de impacto e propagação de mudanças onde dependências transitivas são avaliadas.

O método de grafo de chamadas, conforme ilustrado pela Figura 3.6, é utilizado pela ferramenta JRipples para analisar as dependências. Neste exemplo, são apresentados os tipos de dependência que a classe RecordTimeWorkflowBean possui com outras classes do projeto. É navegando pelo encadeamento dessas dependências que o *grafo de chamadas* é executado.

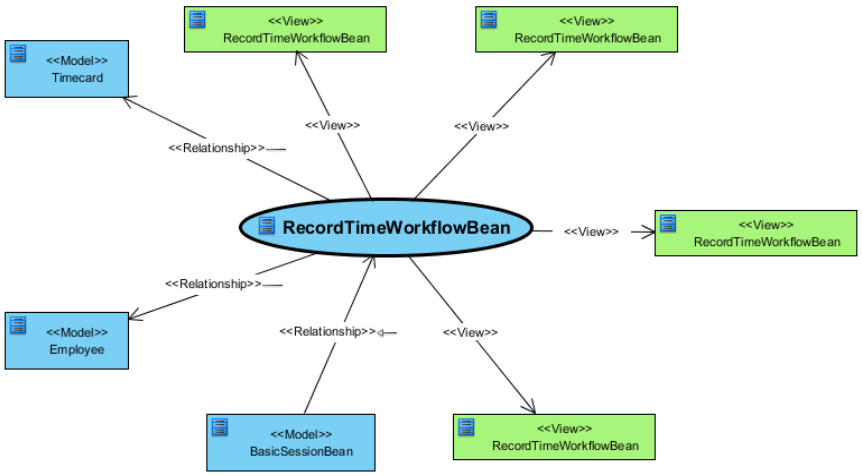


Figura 3.6 – Análise de dependência manual usando JRipples.

A ferramenta JRipples, em oposição às abordagens anteriores, tem cobertura em código e sistematiza a atividade de análise. A ferramenta analisa a dependência dos métodos do programa e solicita ao usuário abrir cada classe dependente indicada pela ferramenta. O usuário abre a classe, analisa o código fonte e decide se é necessária alguma alteração. Caso seja necessário, ele marca na ferramenta a necessidade de revisão e a ferramenta recupera todas as classes relacionadas a este impacto identificado. Caso contrário, a ferramenta ignora e apresenta as próximas classes dependentes. O uso da ferramenta para a análise de impacto está ilustrado em detalhes no Apêndice 2.

[2] http://jripples.sourceforge.net/

Seguindo o exemplo apresentado para a mudança proposta (Figura 3.5), são necessárias alterações na classe de controle *RecordTimeWorkflowBean* e suas interfaces *RecordTimeWorkflow* e *RecordTimeWorkflowHome*, bem como na classe de fronteira *RecordTimeServlet* para validação da data. Para localizar apenas as classes utilizando a ferramenta JRipples, foi necessário analisar 22 diferentes classes (aproximadamente 30%) que equivalem a 1681 linhas de código (aproximadamente 50%), correndo o risco de uma análise equivocada devido à dependência de interpretação do desenvolvedor quanto ao código alterado.

Avaliando as abordagens existentes para a análise de impacto, percebe-se que ainda é uma atividade manual e fortemente baseada na percepção humana, bem como na confiança em uma documentação de apoio.

Conforme apresentado, mudanças são decorrentes de necessidades de negócio. Pela análise do requisito de mudança definido na Figura 3.5, percebe-se que esta necessidade se relaciona a alguns conceitos de domínio apresentados na Figura 3.2. Dentre estes conceitos, pode-se considerar “usuário” (*user*) e “registro de horas” (*time record*), bem como algumas propriedades desses conceitos como “registrar”(*register*) e “horas” (*hours*).

3.2. Modelo de Rastreabilidade

A rastreabilidade vem sendo reconhecida como um fator significante para qualquer fase de um processo de desenvolvimento e manutenção de software [Poh96], contribuindo assim para a qualidade final do produto desenvolvido. Apesar do reconhecimento de sua importância, a rastreabilidade geralmente não é utilizada em escala na indústria de software [Ram01]. A falta de rastreabilidade entre os artefatos pode ser devido a diversos fatores [Zha08a], tais como: o fato que existem artefatos descritos em linguagens diferentes, incluindo linguagem natural, especificação formal, linguagens de programação e modelagem; descreve um sistema computacional em diferentes níveis de abstração; muitos processos não determinam a manutenção de elos de rastreabilidade existentes; e a falta de ferramentas adequadas para apoiar a definição e manutenção de elos de rastreabilidade entre diversos artefatos.

Este fenômeno pode ser resultado da dificuldade de automatizar a geração de relações de rastreabilidade com uma semântica precisa e clara, visto que a maioria dos

ambientes e ferramentas assumem que os elos de rastreabilidade devem ser identificados e mantidos manualmente [Zha08a]. A identificação de elos de rastreabilidade é claramente custosa e não satisfatória [Luc07]. As pesquisas existentes em rastreabilidade de software focam em reduzir o custo e esforço associados à identificação manual através do desenvolvimento de um apoio automático em estabelecer e manter a rastreabilidade entre diferentes artefatos [Zha08a].

Nos últimos anos, ontologias vêm se destacando como o principal formalismo para compartilhar e associar conhecimento entre diferentes modelos [Gua98]. Esta camada de conhecimento habilita o desenvolvimento de sistemas mais flexíveis e adaptativos fazendo uso dos últimos progressos da comunidade de Web Semântica, incluindo OWL. A rastreabilidade orientada a ontologias se refere à efetiva integração de uma camada semântica nas existentes abordagens de desenvolvimento de software.

Ontologias podem nivelar a semântica entre diferentes modelos de software, apoiando os desenvolvedores a encontrar informação relevante entre os diferentes produtos de trabalho, tais como diagramas, casos de teste e código fonte. Neste contexto, a rastreabilidade orientada a ontologias representa a capacidade de rastrear as relações semânticas entre os artefatos ao invés de agrupá-los baseados nos requisitos que motivaram seu desenvolvimento. Com o uso de uma análise de rastreabilidade orientada a ontologias, dois grandes benefícios podem ser atingidos se comparados às demais técnicas de rastreabilidade:

1. Menor granularidade dos elementos rastreados;
2. Identificação de relacionamentos indiretos através da dedução lógica.

Com base em um análise de rastreabilidade orientada a ontologias, é possível associar estruturas de código à conceitos do domínio e, com o uso de técnicas de recuperação de informação e análise de dependência, apoiar a identificação do impacto de determinada mudança.

Com base no apresentado, este trabalho propõe um modelo para rastreabilidade orientada a ontologias a ser utilizada durante o desenvolvimento de software. O objetivo é permitir a identificação, uso e manutenção de relacionamento semântico entre o código fonte e conceitos deste domínio. A abordagem utiliza uma ontologia como o principal artefato para representação do conhecimento. Como existem diferentes tipos de ontologia conforme seu nível de generalidade, tais como ontologias de alto nível, de domínio e de aplicação [Gua98], esta tese sugere o uso de ontologias de aplicação para manutenção do conhecimento. A motivação na escolha se deve que para sistemas de informação

específicos e contextualizados, ontologias de aplicação são utilizadas para representar o papel das entidades da aplicação desempenhado por entidades de domínio.

Ontologias podem manter a semântica entre modelos, ajudando desenvolvedores a encontrar informações relevantes em diferentes produtos de trabalho. Com o objetivo de obter benefícios a partir da rastreabilidade orientada a ontologias, torna-se necessário:

1. definir uma estrutura ontológica genérica capaz de manter elos de rastreabilidade entre conceitos e código fonte da aplicação;
2. definir uma abordagem automatizada para gerar elos rastreabilidade, encontrando as estruturas dos produtos de trabalho associadas a conceitos do domínio.

Para que seja possível utilizar ontologias para rastreabilidade de artefatos, é necessário definir uma abordagem sistemática para projetar e manter uma ontologia de aplicação durante o desenvolvimento de software. A introdução de ontologias no processo de desenvolvimento de software foi apresentada em trabalhos anteriores [Nol07a], cujo escopo foi restrito ao Processo Unificado no que compete a atividades e produtos de trabalho. Este trabalho sugere um novo modelo de processo, desvinculado agora do Processo Unificado e adaptado a um contexto mais genérico de desenvolvimento. Além da evolução deste processo, esta tese propõe novos aspectos relacionados com a população automatizada de ontologias.

O processo de modelagem de conhecimento está descrito em detalhes no Apêndice 3 e é organizado em três fases: Projeto, Manutenção e Verificação. A ontologia é projetada durante a primeira fase, utilizando como base o Modelo de Domínio, representado por um diagrama de classes conceitual. Uma vez que a primeira versão do primeiro modelo de conhecimento é considerada completa, o Engenheiro de Conhecimento [Mae02] estará apto para iniciar a fase de Manutenção. Esta atividade é responsável por manter consistente os elos semânticos entre o conhecimento modelado e os produtos de trabalho do ciclo de vida do software. Cada modificação na ontologia deve ser verificada durante a fase de Verificação. Esta fase consiste em um processo de controle que analisa cada nova versão da ontologia, procurando por inconsistências com suas prévias versões.

Ontologias podem ser utilizadas para diferentes finalidades, visto que capturam o conhecimento produzido durante o desenvolvimento de software. Para o propósito deste trabalho, os conceitos presentes em ontologias serão utilizados para indexar estruturas de código fonte. Uma visão geral do Modelo de Rastreabilidade é apresentado na Figura 3.7.

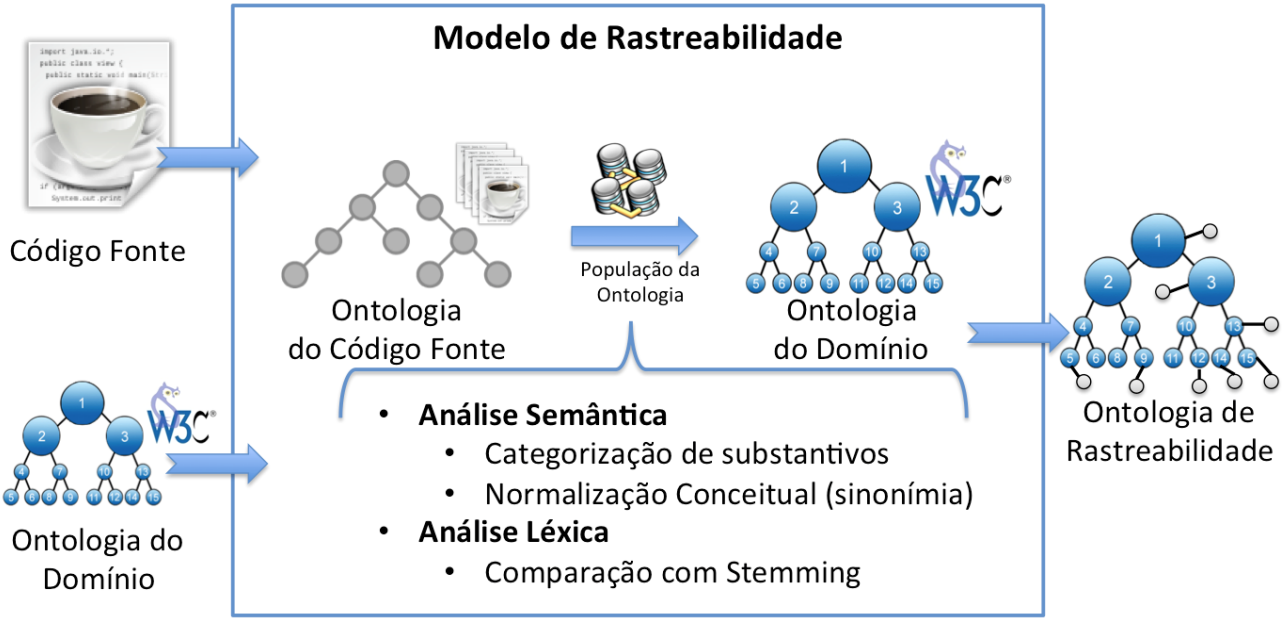


Figura 3.7 – Modelo de Rastreabilidade.

O modelo proposto recebe como entrada o código fonte da aplicação e uma ontologia do domínio. O Modelo de Rastreabilidade realiza a transformação da árvore sintática do código fonte em uma ontologia de código fonte para ser vinculada aos conceitos do domínio. Para a população automática, duas perspectivas foram consideradas: a semântica, que categoriza substantivos, elimina *stopwords*, normaliza os termos utilizando a WordNet e glossário da aplicação e gera *tokens* para comparação; e a léxica, que compara cada *token* utilizando o algoritmo de *stemming*. O resultado desta análise é a ontologia de rastreabilidade, que vincula estruturas do domínio com código fonte.

3.2.1. Modelo de Rastreabilidade Orientado a Ontologias

Linguagens representam formas de comunicação. Como qualquer outra, as linguagens de programação também possuem essa função mantendo duas formas de comunicação: entre humanos e máquinas, através de instruções, e entre humanos e humanos, através de convenções, nomes de identificadores, métodos ou comentários. Para ilustrar esta situação, considere o código ilustrado pela Figura 3.8, que representa um gerador de identificador para todas as classes de domínio da aplicação. Esta classe gera o identificador baseado em um atributo estático concatenado com um valor numérico equivalente a hora corrente.

```
//Id generator for all domain classes
public class IdGenerator
{
  private static int count = 0;
  public static String getId()
  {
    count++;
    long time = new Date().getTime();
    return ""+count+time;
  }
}
```

Figura 3.8 – Exemplo de código com informação semântica.

Quando os nomes da classe, atributos e métodos são substituídos por nomes aleatórios e sem semântica e todos os comentários são removidos, a funcionalidade permanece a mesma sob o ponto de vista da máquina. Contudo, a compreensão humana sobre a funcionalidade ou propósito se torna nebulosa ou mesmo incompreensível. A Figura 3.9 apresenta o mesmo código da Figura 3.8, porém sem informação semântica sobre as estruturas de código.

```
public class Classe
{
  private static int a = 0;
  public static String metodo()
  {
    a++;
    long b = new Date().getTime();
    return ""+a+b;
  }
}
```

Figura 3.9 – Exemplo de código sem informação semântica

Existe muito conhecimento sobre sistemas computacionais que estão associados a conceitos do domínio e são capturados de forma implícita. A maioria dos modelos de rastreabilidade visa formalizar estas relações, associando requisitos a artefatos de software derivados desses requisitos. Quando se discute código fonte, a granularidade associada a métodos e atributos é muito específica para ser gerenciada tendo como base requisitos. Além disso, o conhecimento dos desenvolvedores sobre o domínio é formalizado através da transposição de conceitos em nomes de atributos, métodos e comentários.

Este trabalho assume que uma quantidade razoável de conhecimento sobre o domínio está presente no código fonte da aplicação, relacionando o conhecimento formal da solução com o informal do domínio. O papel de uma ontologia é crucial para formalizar e explicitar este conhecimento, associando-o com estruturas de código. Neste contexto, é proposto um modelo de rastreabilidade para relacionar e recuperar entidades em um

sistema tendo como base conceitos do domínio. Este modelo teve como objetivo desenvolver uma estrutura que seja flexível o suficiente para suportar o mapeamento de quaisquer entidades representadas por artefatos de software, inclusive o código fonte da aplicação. Para tanto, a proposta apresentada em [Nol07a] foi estendida para manter elos de rastreabilidade entre diferentes artefatos e não apenas entre elementos UML.

O recurso “Associacao” é uma classe OWL associada a uma propriedade do tipo objectProperty chamada “ehAssociadaA”. A finalidade dessa propriedade é relacionar elementos de software, tais métodos, atributos, classes, casos de uso, casos de teste, diagramas, etc., com conceitos do domínio, que podem ser classes OWL ou propriedades (*object* e *datatype properties).* Para cada elo de rastreabilidade criado, uma instância (indivíduo) do recurso Associacao é criada ou atualizada, mapeando conceitos do domínio a produtos de trabalho. A estrutura de associação é exemplificada na Figura 3.10 utilizando a notação da [Odm12].

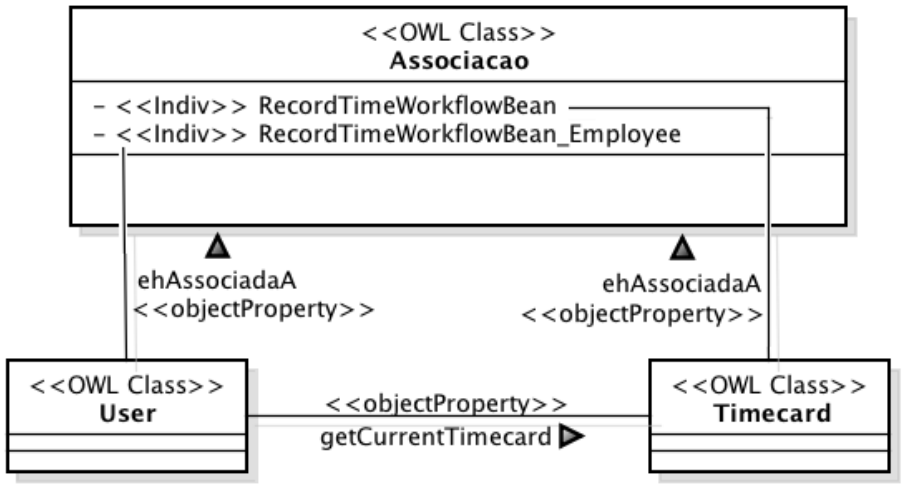


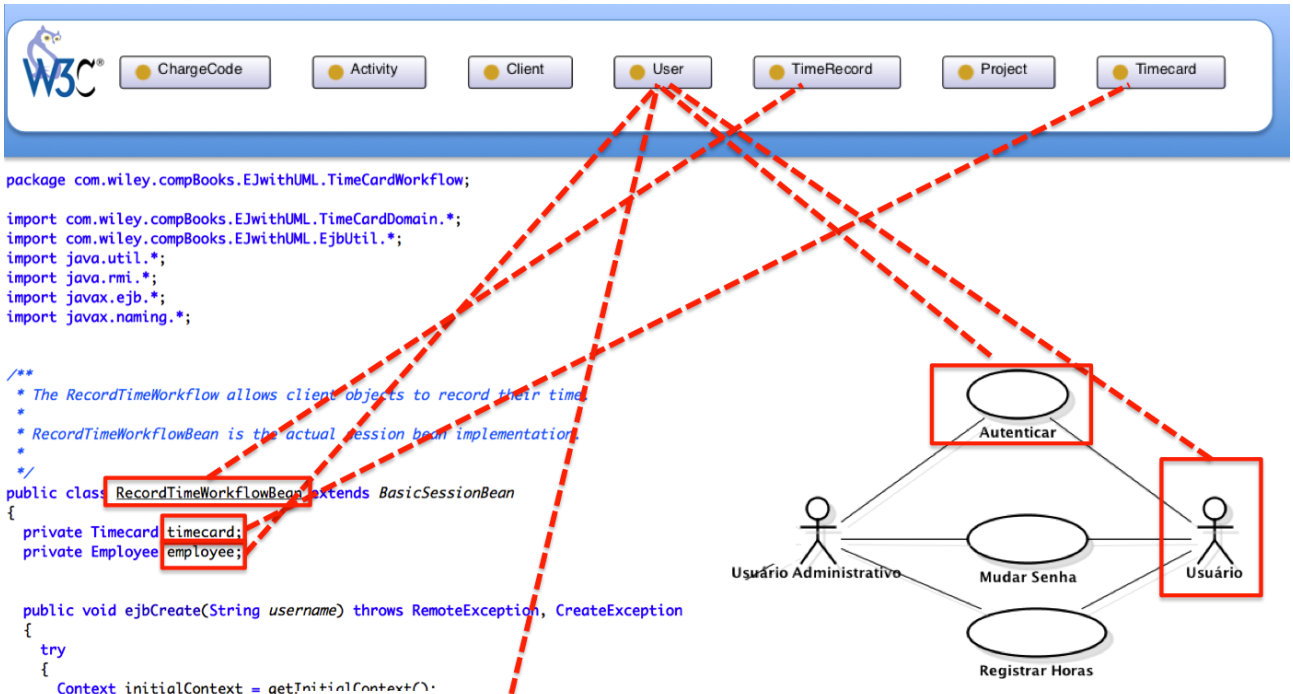
Figura 3.10 – Estrutura ontológica para rastreabilidade entre conceitos e artefatos.

A Figura 3.10 apresenta um exemplo de rastreabilidade obtida através da estrutura Associacao. É ilustrada uma associação entre classe Java RecordTimeWorkflowBean, no caso uma instância da classe Associacao, com o conceito Timecard existente na ontologia de domínio. A figura também ilustra o atributo Employee da mesma classe associado ao conceito da ontologia User. Por sua vez, o conceito User se relaciona com o conceito Timecard na ontologia através da propriedade getCurrentTimecard (objectProperty).

Com esta estrutura, é possível afirmar que ao ser solicitado alguma mudança relacionada ao conceito User, possivelmente a classe Java RecordTimeWorkflowBean será impactada, pois um de seus atributos se relaciona ao conceito User (relacionamento

de dependência). Adicionalmente, o conceito da associação RecordTimeWorkflowBean–Timecard se relaciona indiretamente através de transitividade ao conceito User através da propriedade getCurrentTimecard. Observa-se então que existe uma relação entre a classe Java RecordTimeWorkflowBean e o conceito User, tanto pela dependência de classe e atributo, quanto pelo mapeamento da ontologia. Essa dependência conceitual refletida no código fonte será melhor discutida no Modelo de Probabilidade.

Com esta estrutura, é possível mapear outros artefatos de software, como ilustrado pela Figura 3.11. A ontologia é representada no topo da figura e abaixo diferentes artefatos de software, como código fonte, descrição e diagrama de casos de uso. Este exemplo não é exaustivo, onde poderiam ser apresentados diversos outros diagramas, glossário, documentação de regras de negócio, etc. Também não são apresentados neste exemplo possíveis relacionamentos entre as propriedades OWL com classes da ontologia.



Fluxo de eventos principal: Empregado registra suas horas.	
Passo #	**Ação**
1	O empregado visualiza as horas previamente entradas para o período atual.
2	O empregado seleciona um código de débito dentre os disponíveis, organizados por clientes e projeto.
3	O empregado seleciona um dia do período corrente, que abrange o mês corrente até a data atual.
4	O empregado informa horas trabalhadas como um número decimal positivo.
5	As horas são armazenadas e visualizadas em qualquer acesso subsequente.

Figura 3.11 – Visão geral do modelo de rastreabilidade.

Como o objetivo deste trabalho inclui o mapeamento de conceitos do domínio com o código fonte de uma aplicação, torna-se necessário a utilização de uma estrutura

para representação desse código. Para tanto, o modelo proposto utiliza o conceito de árvore de sintaxe abstrata, ou *Abstract Syntax Tree* (AST), para representar a estrutura do código fonte. Esta árvore representa um construto de um programa [Wel97], isto é, a estrutura sintática e semântica de um código fonte, onde cada nó representa um de seus operadores, tais como operações e atributos, e seus filhos os operandos.

A vantagem em se utilizar uma AST para analisar a estrutura do código, e não uma busca textual simples (leitura de arquivo), é a possibilidade de identificar estruturas bem definidas, pois a AST já traz uma semântica implícita sobre o tipo de construto. Esta estrutura permite identificar, por exemplo, tipos e nomes de atributos, retorno e argumentos de métodos, declaração de variáveis, etc. Com isso, pode-se ignorar *stopwords* como modificadores de visibilidade ou palavras reservadas da linguagem (try, catch, for, private, etc.). A semântica da estrutura do código não poderia ser identificada de forma simples por uma busca textual, a menos que subsidiada por heurísticas.

A abordagem adotada compreende a geração automática de uma ontologia que representa o código fonte da aplicação a partir da AST, dissociada da ontologia do domínio. O objetivo de manipular duas ontologias, uma para código fonte e outra para o domínio, diz respeito diferença de suas naturezas. Existem relações semânticas que fazem sentido no contexto do domínio da aplicação e que não podem ser mapeadas para o código fonte, como, por exemplo, relações de hierarquia de propriedades (*subPropertyOf*), relacionamento de instâncias e classes em um mesmo modelo, etc. Adicionalmente, relações como implementação de interfaces, polimorfismo, padrões de projeto, entre outros, fazem parte da solução da aplicação que, apesar de poderem ser mapeadas a estruturas do domínio, possuem uma organização característica da solução. Com o intuito de manter dissociadas estas estruturas, foi definido o uso de duas ontologias. Apesar de dissociadas, a atividade de população de elos de rastreabilidade entre a ontologia de código fonte e de domínio irá manter os relacionamentos semânticos em uma nova ontologia de rastreabilidade.

O modelo desenvolvido inclui a geração automática da ontologia do código fonte utilizando o JDT (*Java Development Tools*) [Jdt12], que é um *parser* Java fornecido pelo Eclipse [Ecl12]. O sistema analisa a AST do código fonte Java e identifica todas as entidades e seus relacionamentos para gerar a ontologia do código fonte. Este procedimento permite, por exemplo, a identificação de instâncias de classes, isto é, relacionamento entre entidades do código fonte que representam instâncias de outras

classes, através da análise estática. O processo de criação da ontologia do código fonte inclui, para cada classe do projeto, os seguintes procedimentos:

1. Criar uma classe OWL equivalente à classe do código fonte;
2. Para cada atributo da classe do código fonte, criar um datatype property na ontologia subordinado a respectiva classe OWL;
3. Para cada operação da classe do código fonte, criar um object property na ontologia subordinado a respectiva classe OWL.

Um exemplo de uma ontologia do código fonte é ilustrado pela Figura 3.12. Nela existe a taxonomia de classes, datatype properties e object properties. Inicialmente a ontologia do código fonte possuirá as classes da aplicação representadas por classes OWL, operações representadas por *objectProperties* e atributos por *datatypeProperties*. Esta representação é obtida através do mapeamento da AST de cada classe da aplicação, que irá gerar um único modelo OWL com as classes da ontologia equivalentes à aplicação. As relações de referências externas a classes (associações) fazem parte desta ontologia. A ontologia de rastreabilidade e a ontologia do código fonte podem ser visualizadas no Apêndice 4.

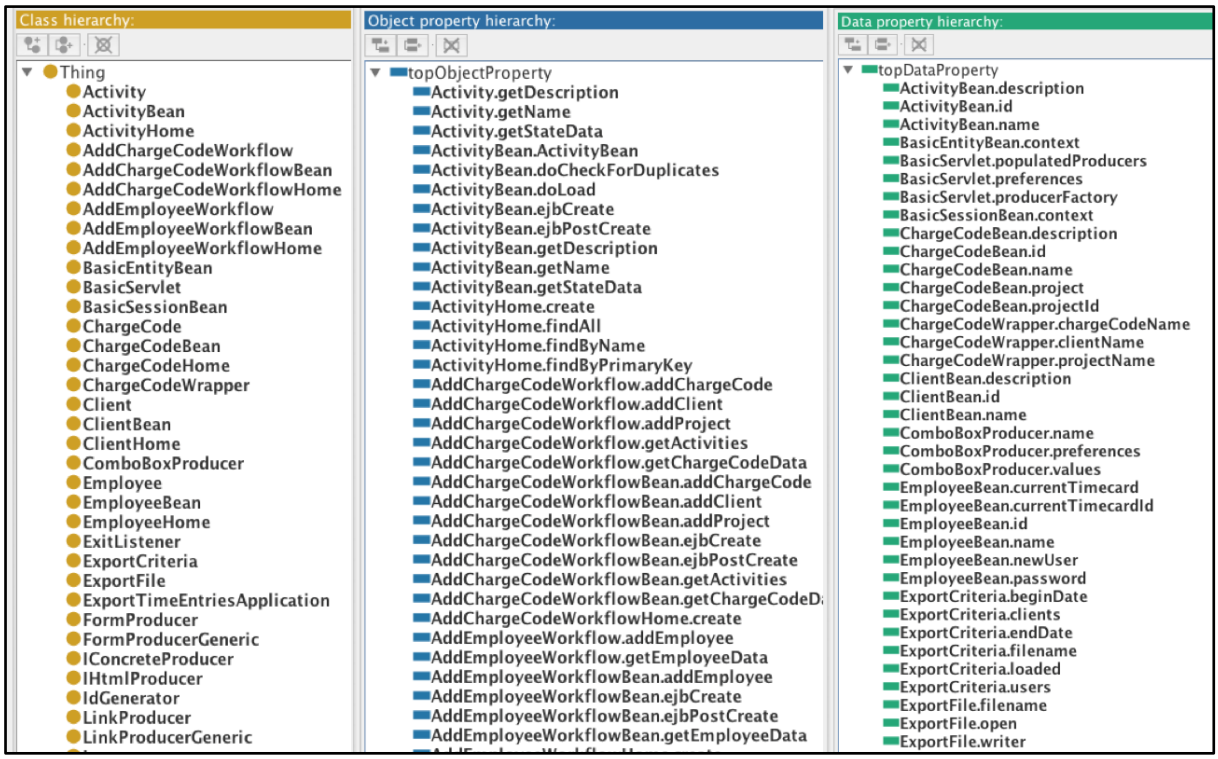


Figura 3.12 – Ontologia do Código Fonte.

Com o modelo de rastreabilidade apresentado, é possível relacionar termos da ontologia do código fonte com os termos da ontologia de domínio. Para tanto, o Modelo

de Domínio, descrito pelo diagrama de classes da UML e apresentado na Figura 3.2, foi traduzido em OWL utilizando a tabela de mapeamento UML para OWL (descrito na Tabela 8.1 do Apêndice 3) e está ilustrado pela Figura 3.13.

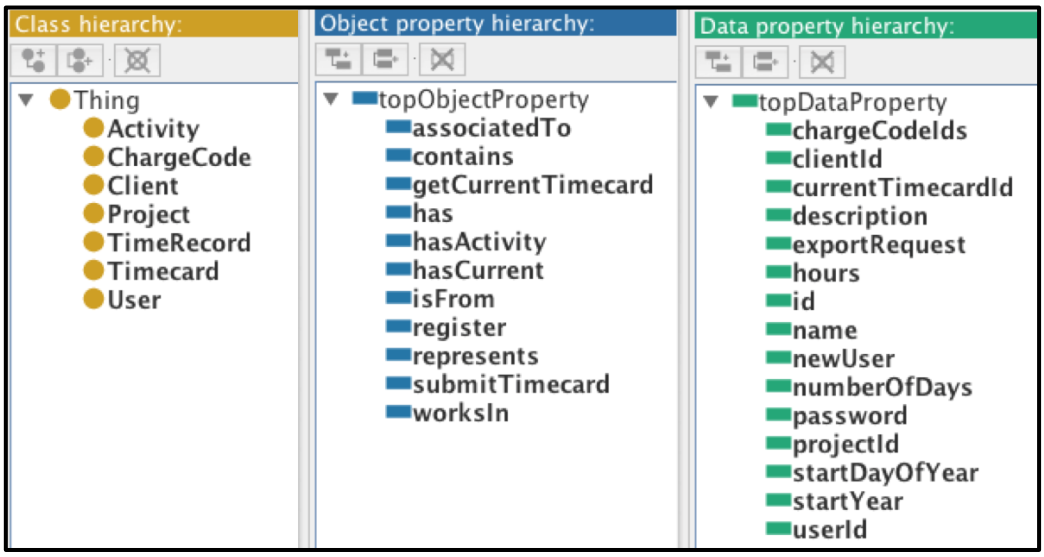


Figura 3.13 – Ontologia de domínio.

O modelo de rastreabilidade desenvolvido relaciona estruturas do domínio com o código fonte conforme apresentado pela Figura 3.14. Esta figura ilustra a rastreabilidade de alguns conceitos e propriedades da ontologia com classes e atributos do código fonte.

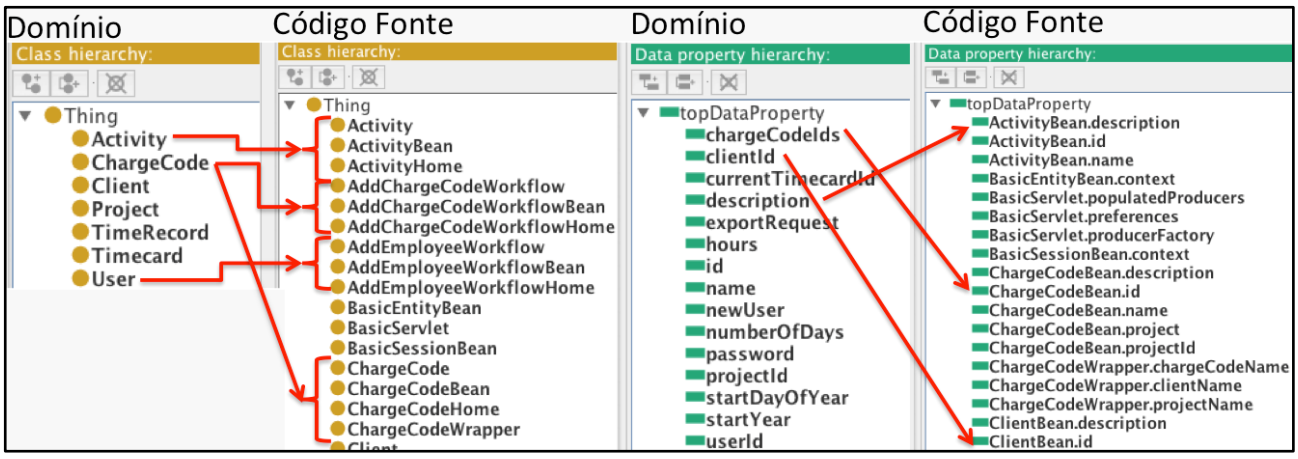


Figura 3.14 – Ontologia de domínio rastreada a ontologia de aplicação.

Como é possível observar na ontologia do código fonte, existem estruturas que não podem ser associadas a conceitos do domínio devido a pertencerem exclusivamente ao escopo da tecnologia empregada como, por exemplo, a classe BasicEntityBean. Na solução apresentada, a ontologia de domínio permanece com sua estrutura inalterada e mantém o conhecimento modelado pelo especialista do domínio (e não da solução),

adicionado apenas elos de rastreabilidade a esta estrutura. Toda a inferência realizada pelo Modelo de Probabilidade será realizada utilizando exclusivamente a ontologia do domínio.

A granularidade da rastreabilidade orientada a ontologias é mais específica se comparada às tradicionais abordagens utilizando requisitos. Essa especificidade possui peculiaridades: quanto mais específico, maior a precisão dos elementos recuperados para um individuo; e quanto mais específico, maior o esforço para manutenção dessas associações.

Para viabilizar o modelo de rastreabilidade, é necessário o apoio de ferramentas para população da ontologia e criação automática de elos de rastreabilidade. Estas ferramentas devem apoiar os desenvolvedores a encontrar novos relacionamentos entre o software e sua especificação, conforme ilustrado previamente. Para tanto, foi desenvolvida uma abordagem baseada em medidas de similaridade entre conceitos da ontologia e especificações de software, apoiando a população da ontologia com indivíduos que mantém os elos de rastreabilidade. Com o objetivo de viabilizar esta proposta, é necessário fornecer ferramentas para esta definição automática. Estas ferramentas podem apoiar os desenvolvedores a encontrar novos conceitos e relações, bem como evoluir a ontologia baseada em produtos de trabalho, incluindo elos de rastreabilidade.

3.2.2. População da Ontologia com Elos de Rastreabilidade

Para apoiar especialistas de domínio e engenheiros do conhecimento na complexa atividade de construir e popular ontologias, algumas técnicas de aprendizagem de máquina e recuperação de informação foram aplicadas para descobrir e explicitar a semântica a partir de dados brutos. O resultado foi descrito em [Nol07a, Nol07d, Nol08] e compreende um modelo e ferramenta para popular uma ontologia de aplicação, capaz de relacionar os conceitos do domínio com estruturas do código fonte.

Aprendizagem de ontologias explora um conjunto de recursos existentes, tais como textos, tesauros, dicionários ou banco de dados, para desenvolver uma ontologia de uma forma semiautomática [Mae02]. O processo de aprendizagem que estende conceitos da ontologia é chamado de População da Ontologia e é realizado pela seleção de fragmentos de texto, associando aos mesmos conceitos da ontologia [Cim06]. Neste

contexto, o mapeamento linguístico é discutido na literatura como sendo um processo de três passos [Mad01]:

1. *categorização*: dados são agrupados em uma ou mais categorias nas quais a similaridade é comparada. Esse passo reduz o número de comparações unitárias, considerando apenas elementos dentro da mesma categoria linguística;
2. *normalização*: dados similares em diferentes modelos podem possuir nomes lexicamente diferentes devido ao uso de abreviaturas, convenções, pontuações ou sinônimos. Como parte da normalização, é realizada a geração de tokens (reduzindo o termo à sua raiz morfológica), expansão (recuperando sinônimos, convenções e abreviaturas) e eliminação (ignorando palavras não relevantes ao contexto como preposições, artigos e outras). Em cada um desses passos, um tesauro é utilizado para relacionar termos comuns da linguagem e referências específicas do domínio a partir do produto de trabalho Glossário;
3. *comparação*: o coeficiente de similaridade linguística é computado entre o nome dos elementos normalizados em cada categoria, incluindo a recuperação do significado semântico e avaliação parcial do termo (substring).

Além dos três passos descritos, existem diferentes níveis para medir a similaridade entre os termos. Esta proposta utiliza uma visão em duas camadas, conforme sugerido por [Mae02], sendo elas: léxica, que investiga como os termos se relacionam ao significado, e semântica, que investiga as relações conceituais entre os termos.

3.2.2.1. Análise de Similaridade Léxica

Dois métodos de comparação bem estabelecidos são Distância de Edição [Lev66], para ponderar a diferença entre duas palavras medindo o número mínimo de caracteres a serem inseridos, excluídos ou atualizados para transformar uma palavra em outra; e Stemmer [Lov68], para redução da palavra a sua forma canônica, removendo flexões ou derivações a partir da sua raiz morfológica.

O trabalho de [Kan00] apresenta um estudo que avalia o algoritmo de Stemmer. Três diferentes métodos foram comparados, incluindo Distância de Edição, Distância de Edição com Edição Complexa e o Prefixo Stemmer. O corpora utilizado para avaliação dos métodos foi o das palavras mais comuns do ano de 2000 da *Associated Press and*

Reuter's. Apesar da tendência do método de Distância de Edição possuir um melhor desempenho em palavras com erros de digitação e tipográficos, o estudo experimental evidenciou que o Prefixo Stemmer é mais preciso que os demais métodos.

Associado a isso, em um típico cenário de desenvolvimento, no qual cada produto de trabalho passa por diferentes revisões e verificações evitando erros tipográficos, a ideia da utilização do algoritmo de Stemmer é natural. Baseado nestas evidências, esta proposta utilizou o método de Stemmer para comparação léxica.

3.2.2.2. Análise de Similaridade Semântica

O nível de comparação semântico avalia as relações conceituais entre os termos. Esta análise considera o mapeamento entre conceitos e não o mapeamento entre termos, realizada pela análise léxica. O mapeamento entre conceitos consiste em analisar o significado das palavras através de suas relações semânticas obtidas durante a normalização, permitindo a recuperação de termos mesmo com grafias diferentes. A normalização utiliza um tesauro que avalia as relações semânticas e léxicas entre os termos. A [Wor12] representa uma grande base de dados léxica que contém verbos, substantivos, adjetivos e advérbios organizados em categorias. Estas categorias são relacionadas por sentido de relações léxicas e semânticas. O resultado é uma rede que associa palavras aos seus conceitos. A análise de similaridade utiliza a WordNet para recuperação de relações de sinonímia entre termos. Para cada termo na especificação de software, é analisada a similaridade léxica com os conceitos da ontologia e, caso não identifique, é realizada a mesma análise com os termos sinônimos. Além da WordNet, é realizada a busca de sinônimos em um glossário da aplicação.

3.2.2.3. Análise de Similaridade e População de Ontologias

A solução desenvolvida para a população automática dos elos de rastreabilidade entre ontologia e estruturas do código fonte inclui a categorização, normalização e comparação léxicas e semânticas, ilustrada pela Figura 3.15. Como a proposta tem o objetivo de identificar os conceitos do domínio a partir do código fonte do programa, apenas substantivos são considerados para as categorias linguísticas.

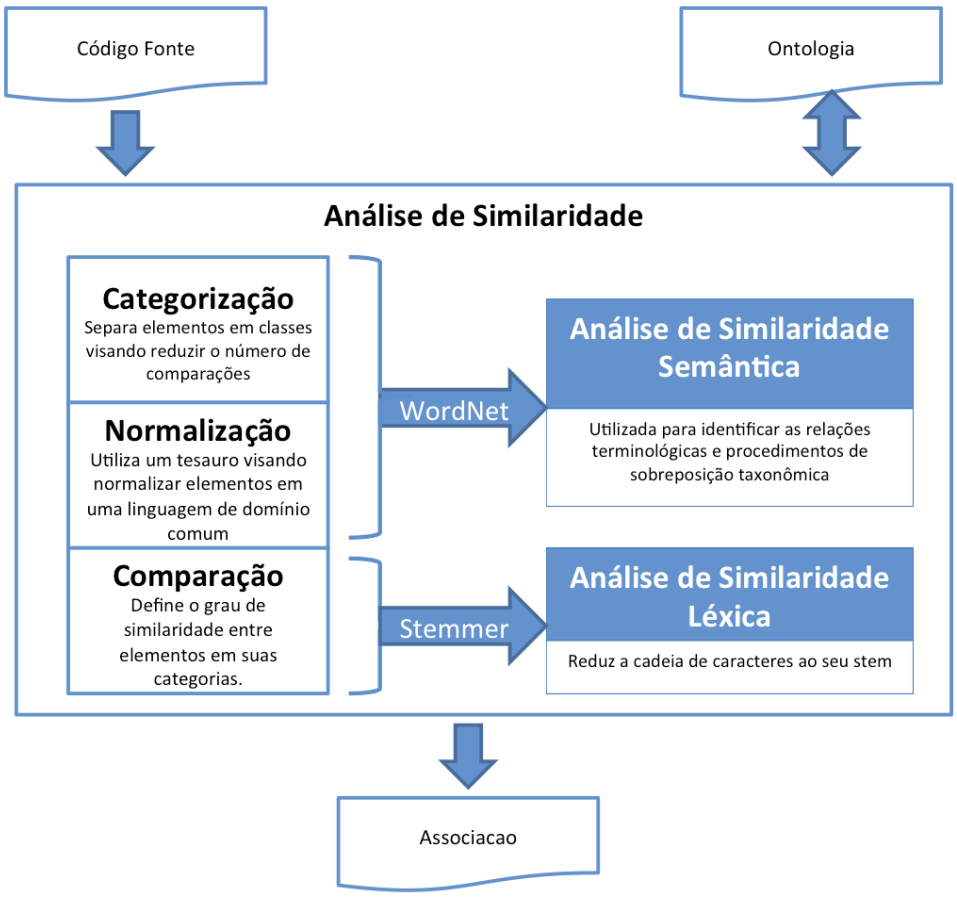


Figura 3.15 – População automática de elos de rastreabilidade.

O fluxo de processo inclui a avaliação de cada elemento da especificação de software e seus sinônimos, considerando apenas aqueles pertencentes à categoria linguística de substantivos. A avaliação inclui uma análise léxica de cada elemento da especificação com os recursos da ontologia utilizando o algoritmo de Stemmer. Em caso de similaridade, este relacionamento é adicionado na matriz de rastreabilidade da ontologia. Se os termos não são lexicamente equivalentes, o próximo passo é normalizar o termo, reduzindo a redundância. A normalização inclui a recuperação de termos sinônimos na WordNet e glossário para execução da análise léxica previamente descrita. A busca por sinônimos não é cíclica e é realizada uma única vez para cada elemento analisado.

A população da ontologia para os elos de rastreabilidade irá comparar cada termo categorizado como substantivo no código fonte (também considerando individualmente nomes compostos) com todas as classes e propriedades da ontologia. Caso a raiz morfológica (*stem*) dos termos seja equivalente, o sistema cria uma instância da classe OWL Associação, relacionando a estrutura do código com o elemento da ontologia. Essa análise deve considerar cada componente de palavras compostas, separadas por

caracteres maiúsculos, números e *underscore*. Se os termos não são equivalentes, o sistema irá buscar na WordNet e glossário uma lista de sinônimos do termo e analisar com todos os conceitos da ontologia. O algoritmo que representa este processo é apresentado na Figura 3.16.

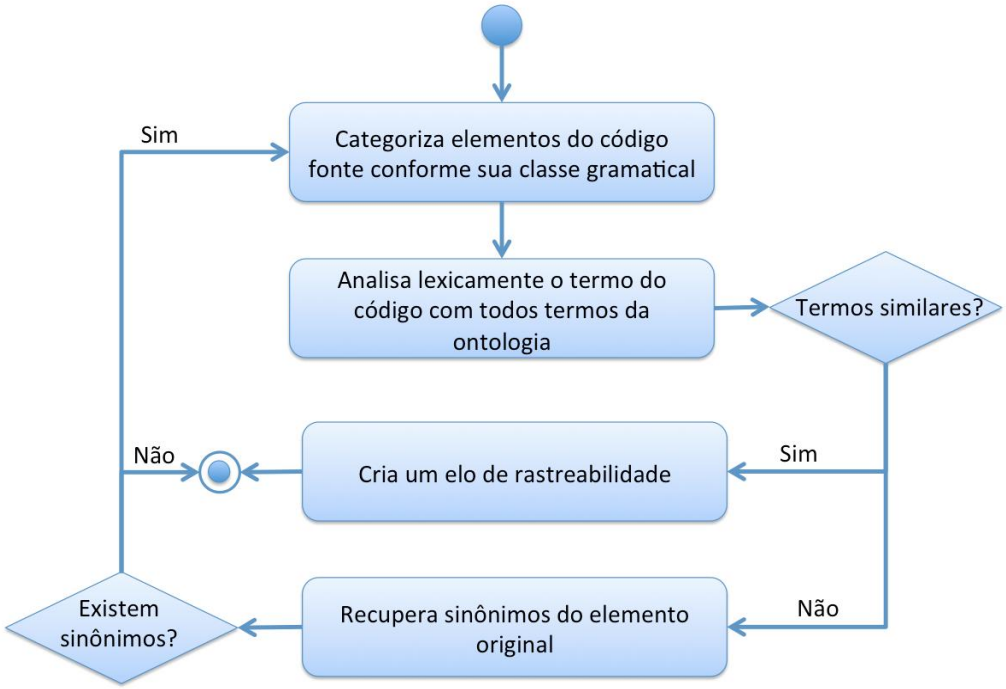


Figura 3.16 – Processo para análise de similaridade automática.

Como exemplo de uso deste algoritmo, sugere-se considerar o termo da ontologia de domínio “user”. Comparando lexicamente este termo, cujo stem é “user”, com a estrutura de código Java “Employee”, cujo *stem* é "empl”, percebe-se que não existe similaridade.

Ao recuperar da WordNet o sinônimo (*Synset Semantic Relation*) do termo “user”, obtém-se: “user (a person who makes use of a thing; someone who uses or employs something)”. O termo “employs” será primeiro categorizado quanto a sua classe gramatical. Recuperando da WordNet, o mesmo possui o estado de substantivo (noun.state) como “employment” e “employ”, podendo então ser candidato a comparação léxica.

Analisando os *stems*, identifica-se a equivalência léxica entre o termo “employs”, que é sinônimo do conceito “user”, com a referência do código Java “Employee”. Com isso, gera-se um elo de rastreabilidade entre “user” (domínio) e “Employee” (código fonte).

O modelo de rastreabilidade gera como resultado uma matriz que relaciona conceitos a suas estruturas de código, isto é, um grafo direcionado dos conceitos do

domínio para as estruturas de código. Para gerar tal matriz, deve-se percorrer todas as classes do código fonte da aplicação e verificar se existe equivalência com classes OWL e, para cada classe da aplicação, verificar se seus atributos e métodos possuem equivalência com datatype properties e object properties, respectivamente. A granularidade dos elos deve considerar classes e todos os seus membros. A Tabela 3.3 apresenta os procedimentos associados a cada estrutura da classe. Como resultado, se houver equivalência, a ontologia de domínio é populada com o elo de rastreabilidade, vinculando o conceito a sua estrutura de código.

Tabela 3.3 – Análise de rastreabilidade entre código fonte e ontologia.

Código Fonte	**Ontologia de Domínio**
Nome da Classe	Realizar a análise de similaridade com todas as classes OWL.
Declaração do atributo (nome e tipo)	Realizar a análise de similaridade com todos datatype properties.
Assinatura do método (nome, retorno e argumentos)	Realizar a análise de similaridade com todos object properties.

A Figura 3.17 apresenta um exemplo do modelo de rastreabilidade relacionando respectivamente a classe OWL *ChargeCode*, datatype property *day* da classe *Entry* e object property *get_hours* da classe *User* e *Entry* com suas estruturas definidas pela ontologia do código fonte (Figura 3.18). As ontologias de domínio populada com os elos de rastreabilidade e de código fonte estão disponíveis no Apêndice 4.

```
  <rdf:Description rdf:about="http://www.owl-
ontologies.com/Ontology1210823302.owl#ChargeCode">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/Semantics/SourceCode#ChargeCodeWrapper"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowHome"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowBean"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflow"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeHome"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCode"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.owl-ontologies.com/Ontology1210823302.owl#day">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="http://www.owl-
ontologies.com/Ontology1210823302.owl#Entry"/>
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.numberOfDays"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCodeTimecardBean.startDayOfYear"/>
  </rdf:Description>
```

```
  <rdf:Description rdf:about="http://www.owl-
ontologies.com/Ontology1210823302.owl#get_hours">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="http://www.owl-
ontologies.com/Ontology1210823302.owl#User"/>
    <rdfs:range rdf:resource="http://www.owl-
ontologies.com/Ontology1210823302.owl#Entry"/>
    <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflow.getHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflow.setHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.setHours"/
>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.getHours"/
>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeServlet.extractHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.setHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.getHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.storeHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.loadHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Timecard.getHours"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Timecard.setHours"/>
</rdf:Description>
```

Figura 3.17 – Ontologia de rastreabilidade.

```
<rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowHome">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.numberOfDays">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.setHours">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Description>
```

Figura 3.18 – Ontologia parcial do código fonte.

3.3. Modelo Probabilístico

O Modelo de Rastreabilidade define uma relação de dependência entre conceitos do domínio e o código fonte da aplicação através de uma ontologia de rastreabilidade. Esta relação pode ser representada por um modelo de recuperação de informação booleano, considerando apenas a presença de índices dos termos (conceitos) no código

fonte da aplicação. Com este modelo, é possível identificar se uma classe ou seu membro é relevante para a busca fazendo uso de álgebra booleana. Como consequência, todas as classes que estiverem vinculadas a um conceito serão recuperadas por esta busca. O modelo booleano não fornece os meios para quantificar a influência de determinado grupo de conceitos, ou suas propriedades, no código da aplicação. Analisando o modelo vetorial, os índices dos termos (conceitos, no caso) são assumidos como independentes uns dos outros, não considerando seu contexto semântico e a dependência conceitual entre as estruturas de código, um conhecimento valioso que pode ser representado em uma ontologia. Para avaliar essa influência e diagnosticar o impacto de uma mudança de forma quantitativa, foi escolhido utilizar o modelo de raciocínio probabilístico de Redes de Crenças Bayesianas (RCB).

RCB representam um método bem estabelecido na comunidade de Inteligência Artificial para raciocínio em situações de incerteza, utilizando (1) uma estrutura de grafo direcionado representando relacionamentos de causa e efeito e (2) um cálculo de probabilidade para quantificar essas relações, atualizando as crenças dada uma nova informação. Esta rede representa a união de uma distribuição de probabilidade e um conjunto de variáveis, consistindo em uma parte qualitativa (grafo direcionado) e outra quantitativa (modelos de dependência). Para um domínio em particular, os vértices do grafo representam variáveis enquanto as arestas direcionadas descrevem as probabilidades de transição entre estas variáveis.

Utilizando um modelo RCB associado a uma matriz de rastreabilidade, viabiliza-se a predição do impacto de uma mudança em um sistema quando certos conceitos da aplicação são identificados. A RCB é um grafo cujas arestas conectam os nodos sem formar ciclos diretos, isto é, um grafo acíclico direcionado do conceito para o código fonte. Os nodos representam variáveis aleatórias e as arestas representam a dependência direta entre essas variáveis. Este trabalho assume como dependência os elos de rastreabilidade entre estruturas de código e conceitos da ontologia.

Para ilustrar melhor o modelo proposto, a Figura 3.19 apresenta uma visão geral. Como entrada deste modelo, a matriz de rastreabilidade, representada pela ontologia de rastreabilidade, e um requisito de mudança são fornecidos. O requisito de mudança é então analisado léxica e semanticamente em busca de referência a conceitos do domínio. Uma vez identificados, os mesmos são utilizados para rastrear as estruturas do código fonte da aplicação e realizar o cálculo de relevância. Este cálculo é realizado utilizado RCB incluindo duas análises distintas: probabilidade independente de cada conceito da

aplicação e código fonte, utilizando o algoritmo de classificação PageRank, e probabilidade dependente, utilizando o algoritmo de frequência TFIDF e análise da dependência conceitual fazendo uso de grafos de chamada.

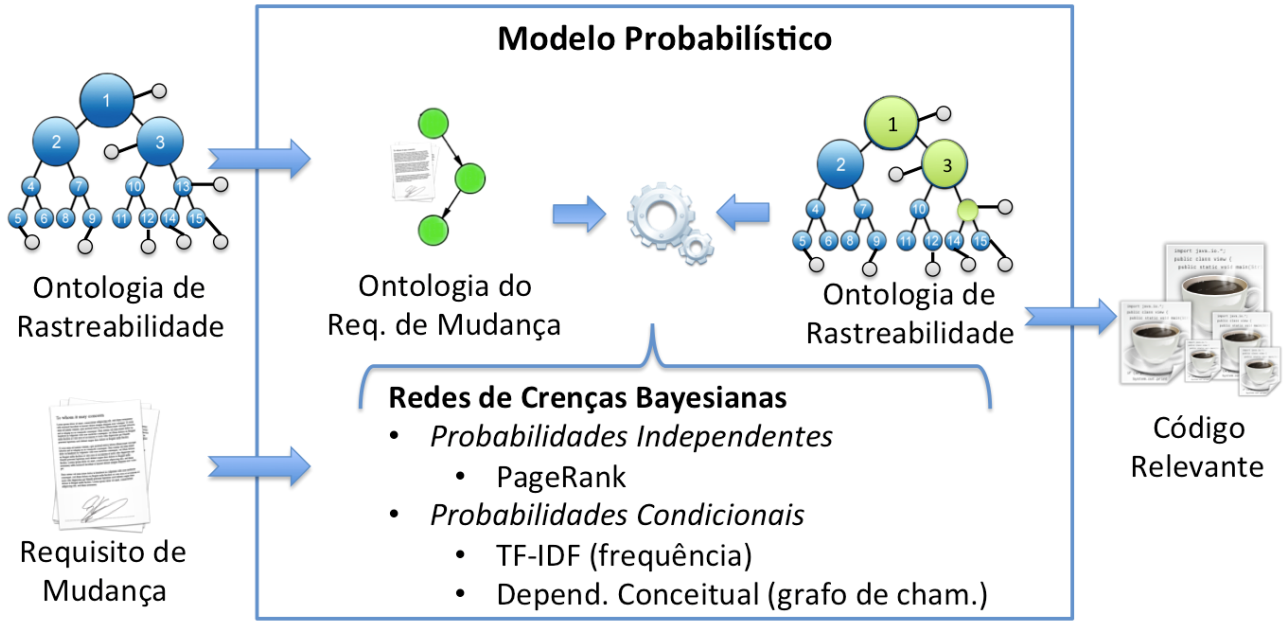


Figura 3.19 – Modelo de Probabilidade.

Segundo [Kor04], a definição de uma RCB para um domínio particular envolve as seguintes tarefas: (1) identificação de variáveis relevantes e seus valores, equivalendo aos nodos e seus estados; (2) identificação e representação do relacionamento entre estes nodos (o que significa quais nodos devem ser conectados); e (3) a parametrização da rede, que consiste na (a) definição de probabilidades iniciais para cada nodo e (b) em probabilidades condicionais associadas a cada aresta.

3.3.1. Definição de Nodos e Estados

Na teoria, variáveis podem ser discretas ou contínuas. Os modelos criados nesse trabalho foram construídos utilizando exclusivamente variáveis discretas, isto é, o espaço amostral se restringe a um conjunto finito de classes e propriedades das ontologias. É importante relembrar que este trabalho utiliza dois conjuntos distintos de variáveis que equivalem ao domínio e contradomínio dos elos de rastreabilidade. O domínio (D) consiste no conjunto finito de conceitos do domínio do problema e o contradomínio (CD) os recursos extraídos a partir do código fonte da aplicação. A imagem (Im) representa, neste contexto, todas as estruturas do código fonte que foram rastreadas a conceitos. Para ilustrar essa situação, considere a Figura 3.20, onde:

$$D = \{\text{Conc01}, \text{Conc02}, \text{Conc03}, \text{Obj. Pro01}, \text{Data. Pro01}\}$$

$$CD = \{\text{Class01}, \text{Class02}, \text{Class03}, \text{Class04}, \text{Método01}, \text{Atributo01}\}$$

$$Im = \{\text{Class01}, \text{Class02}, \text{Attributo01}\}$$

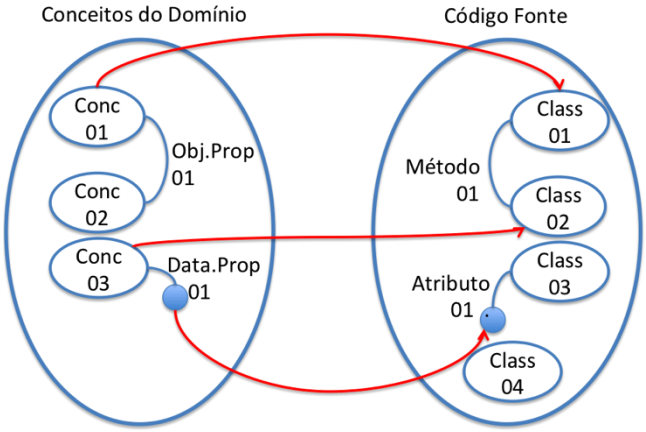


Figura 3.20 – Identificação de nodos e estados.

Cada nodo possui uma probabilidade condicional calculada dinamicamente conforme os conceitos identificados pelo requisito de mudança, quantificando assim a relevância de cada elo de rastreabilidade. Caso sejam identificados alguns conceitos referentes a uma solicitação de mudança, então é provável que as estruturas de código dependentes se tornem instáveis e suscetíveis a mudanças. Antes de representar esta dependência, é importante representar os possíveis estados que os conceitos do domínio e código fonte podem assumir. Para tanto, foi definido que ambos os conjuntos possuem dois estados que representam quando um nodo é estável ou instável (possivelmente será modificado devido a solicitação). Os dois estados que um nodo pode assumir são:

- **Estável (E)**: o nodo não é identificado como impactado;
- **Instável (I)**: o nodo é identificado como passível de mudança.

Para ilustrar, considere que na Figura 3.20 apenas o conceito Conc01 foi identificado como instável, então apenas a classe Class01 será considerada como instável.

3.3.2. Definição de Arestas

Cada elo de rastreabilidade é representado pela associação de um nodo pertencente ao domínio com um nodo pertencente à imagem. Essa organização foi definida para prevenir ciclos, fazendo com que RCB pudesse ser aplicado de forma

válida, isto é, em um grafo acíclico direcionado. Quando o modelo de rastreabilidade é formado, ele é sempre baseado em nodos interconectados com a estrutura fundamental: $D \rightarrow Im$.

Esta estrutura representa as relações direcionadas a partir dos conceitos do domínio para o código fonte da aplicação. Estas relações podem definir diferentes situações:

- Um conceito x é considerado estável se não for identificado em uma requisição de mudança;
- Um conceito x é considerado instável se for identificado em uma requisição de mudança;
- Um código fonte y é considerado estável se y pertence ao conjunto Im e não existe nenhum conceito x instável pertencente a D que seja rastreável para y ou se Y pertence ao CD e não pertence a Im;
- Um código fonte y é considerado instável se e somente se y pertence ao conjunto Im e existe algum conceito x instável pertencente a D que seja rastreável para y.

As relações são definidas e quantificadas utilizando uma matriz de probabilidade baseada em nodos de D, Im e suas relações.

3.3.3. Definição de Probabilidades dos Relacionamentos

A intenção em usar RCB é definir a relevância de uma estrutura de código em particular baseado em um requisito de mudança. Para tanto, a RCB provê uma probabilidade condicional de observar um evento y dado x – uma probabilidade que o evento y ocorra (código potencialmente impactado) uma vez que o evento x tenha ocorrido (conceito identificado como relevante). Em particular, essa classificação utiliza o seguinte:

- $P(x)$: a probabilidade de observar o evento x, também conhecida como probabilidade inicial ou marginal;
- $P(y)$: a probabilidade de observar o evento y, também conhecida como evidência ou probabilidade marginal de y, agindo como uma constante normalizadora.

- $P(y|x)$: a probabilidade de observar o evento y caso ocorra o evento o x, também conhecido como probabilidade condicional;

A parte essencial de um classificador é o cálculo da probabilidade observada de um evento y relacionado ao evento x, também conhecida como probabilidade posterior e referenciada por $P(x|y)$. O cálculo é realizado pelo teorema de Bayes:

$$P(x|y) = \frac{P(y,x) \times P(x)}{P(y)}$$

O classificador de NaiveBayes pode fornecer uma medida de quão provável uma estrutura de código y será impactada quando submetida uma consulta incluindo o conceito x. Este classificador não é utilizado para efetivamente classificar, mas produzir uma medida de relevância na análise do impacto. Para tanto, serão discutidas as probabilidades marginal e condicional.

3.3.3.1. Cálculo da Probabilidade Marginal P(x)

Cada nodo raiz possui uma probabilidade previamente definida como $P(x)$. Estabeleceu-se que cada nodo raiz é definido por conceitos da ontologia x, tal que x pertence a D. A arquitetura definida para o modelo de probabilidade deve estimar e atribuir as probabilidades para cada um dos nodos, que indica o quão relevantes são os conceitos entre si, independente de suas dependências com o código fonte (y).

Para calcular a probabilidade marginal de forma independente de y, optou-se pelo algoritmo de classificação PageRank, originalmente utilizado pelo motor de busca do Google. Este algoritmo avalia a relevância, ou popularidade, de um nodo de um grafo com base nas dependências atribuídas ao mesmo. Maiores detalhes podem ser obtidos no Capítulo 2.

Assumindo que classes nas ontologias são associadas por propriedades do tipo object properties e que relacionam domínio e imagem, é possível identificar a relevância de um conceito dentro do seu universo pela quantidade de conceitos que o referencia e que são referenciados por ele. Essa medida fornece a probabilidade na qual um determinado conceito pode ser modificado baseado em suas dependências.

Para a obtenção da relevância usando PageRank, primeiro é necessário definir o grafo no qual esta relevância será extraída. Percorre-se então todas as object properties da ontologia e inclui-se no grafo do PageRank uma aresta cuja origem é o domínio

(*domain*) da propriedade e destino sua imagem (*range*). O valor alfa (ajuste de primitiva) utilizado para o algoritmo é 0.14 conforme se estima ser o valor utilizado pelo Google [Mar09]. Observando a ontologia apresentada na Figura 3.13, o resultado da aplicação do PageRank pode ser observado na Tabela 3.4.

Tabela 3.4 – PageRank para os conceitos da ontologia.

Conceito	**PageRank**
User	0,223961857
Timecard	0,151126526
TimeRecord	0,146297665
Activity	0,145177449
Client	0,138264987
ChargeCode	0,098145865
Project	0,09702565

Os valores apresentados na Tabela 3.4 correspondem a probabilidade independente p(x) que representa o quão importante um conceito é em seu domínio.

3.3.3.2. Cálculo da Probabilidade Marginal P(y)

O algoritmo de classificação PageRank também foi utilizado para o cálculo da probabilidade marginal de y, utilizando o mesmo valor alfa descrito anteriormente. Como o código fonte também é representado por uma AST, o grafo do código fonte é gerado navegando por esta AST e identificando todos os tipos utilizados por cada classe ou interface do código fonte. Toda a análise de P(y) é independente dos conceitos da ontologia.

Para a geração do grafo necessário ao PageRank, para cada classe do código fonte do projeto, verifica-se:

- *Cada atributo* de instância ou de classe (estático), verificando se o seu tipo pertence a alguma classe do projeto.
- *Cada método*, verificando:
    1. Assinatura, avaliando tipo de retorno e tipo dos argumentos; e
    2. Corpo, avaliando a declaração de atributos em (a) blocos de exceções (try-catch), (b) laços (while, for), (c) condicionais (if) e (d) todos os subblocos do método, recursivamente.

A comparação entre os tipos encontrados é léxica entre o elemento procurado e todas as classes existentes no projeto. Para cada tipo encontrado, é criado uma associação entre a classe base e aquela referenciada por ela (e definida pelo tipo contido nela). Este procedimento só considera os tipos existentes entre as classes do projeto em questão, ignorando tipos de bibliotecas como, por exemplo em Java, tipos ArrayList, String, etc.

Para ilustrar, considere o código parcial da classe EmployeeBean ilustrado pela Figura 3.21.

```
public class EmployeeBean extends BasicEntityBean
{
  public String id;
  public String currentTimecardId;
  public String name;
  public String password;
  public boolean newUser;
  private Timecard currentTimecard;

  public String ejbCreate(String name, String password) throws
RemoteException, CreateException
  {
    this.id = "Employee" +IdGenerator.getId();
    this.name = name;
    this.password = password;
    this.newUser = true;

    return null;
  }

  public Timecard getCurrentTimecard()
  {
    try
    {
      if (this.currentTimecard == null)
      {
        Context initialContext = getInitialContext();
        TimecardHome thome =
(TimecardHome)initialContext.lookup("TimecardBean");
        this.currentTimecard =
thome.findByPrimaryKey(this.currentTimecardId);
      }
    }
    catch (Exception e)
    {
      throw new Exception(e.toString());
    }
    return this.currentTimecard;
  }
}
```

Figura 3.21 – Código fonte parcial da classe EmployeeBean.

A Figura 3.21 ilustra um trecho de código que irá gerar uma aresta no grafo de cálculo do PageRank da classe EmployeeBean para as classes sublinhadas na figura

BasicEntityBean, Timecard e TimecardHome. O resultado do cálculo de relevância de todo o código fonte se encontra na Tabela 3.5.

Tabela 3.5 – PageRank para o código fonte da aplicação.

Classe do Código Fonte	PageRank	Classe do Código Fonte	PageRank
Activity	0,021764696	MenuButtonProducer	0,008479239
ActivityBean	0,007627356	Node	0,021368917
ActivityHome	0,015228076	PageProducer	0,013815361
AddChargeCodeWorkflow	0,014767868	PageProducerGeneric	0,007939714
AddChargeCodeWorkflowBean	0,007627356	ProducerFactory	0,013867866
AddChargeCodeWorkflowHome	0,007939714	Project	0,080660182
AddEmployeeWorkflow	0,014186882	ProjectBean	0,007627356
AddEmployeeWorkflowBean	0,007627356	ProjectHome	0,01257303
AddEmployeeWorkflowHome	0,007627356	RadioButtonProducer	0,008479239
BasicServlet	0,007627356	RecordTimeServlet	0,007627356
ChargeCode	0,064815058	RecordTimeWorkflow	0,015639348
ChargeCodeBean	0,007627356	RecordTimeWorkflowBean	0,007627356
ChargeCodeHome	0,01257303	RecordTimeWorkflowHome	0,008408252
ChargeCodeWrapper	0,026927668	SelectableTableDataProducer	0,008479239
Client	0,096170509	SubmitButtonProducer	0,010581
ClientBean	0,007627356	SubmitButtonProducerGeneric	0,007627356
ClientHome	0,015852793	TableProducer	0,010893359
ComboBoxProducer	0,008479239	TableProducerGeneric	0,007939714
Employee	0,029591119	TabularInputFormProducer	0,011394584
EmployeeBean	0,007627356	TabularInputFormProducerGeneric	0,007627356
EmployeeHome	0,017672517	Test	0,007627356
ExportCriteria	0,008223676	TestHttpServletRequest	0,007627356
ExportTimeEntriesApplication	0,007627356	TestLogin	0,007627356
FormProducer	0,010581	TestServlet	0,007627356
FormProducerGeneric	0,007627356	TextEntryFrame	0,007627356
IConcreteProducer	0,008479239	TextFieldProducer	0,010581
IHtmlProducer	0,05928314	TextFieldProducerGeneric	0,007627356
LinkProducer	0,009208075	TextProducer	0,013262076
LinkProducerGeneric	0,007627356	TextProducerGeneric	0,007939714
LoginServlet	0,007627356	Timecard	0,050204174
LoginWorkflow	0,016123504	TimecardBean	0,007627356
LoginWorkflowBean	0,007627356	TimecardHome	0,012050066
LoginWorkflowHome	0,00866855	ExportFile	0,008223676

Os valores apresentados na Tabela 3.5 correspondem a probabilidade independente p(y) que representa o quão importante uma classe é em seu projeto.

3.3.3.3. Cálculo da Probabilidade Condicional P(y|x)

Duas análises complementares foram definidas para o cálculo de probabilidade condicional das classes do código fonte dado um conceito: TFIDF e análise de Dependência Conceitual (DC). A medida TFIDF é uma análise que verifica a ocorrência do conceito em todas as estruturas da classe, enquanto a DC representa uma análise da distância entre os conceitos impactados comparada com a distância entre as classes da aplicação associadas a estes conceitos. O cálculo da probabilidade condicional é dado pela média geométrica das duas medidas devido a pertencerem a amostras diferentes e definido conforme:

$$P(y|x) = \sqrt{TFIDF(y|x) \times DC(y|x)}$$

O algoritmo TFIDF descrito no Capítulo 2 foi utilizado para calcular a probabilidade condicional de alterar a classe do código fonte y caso o conceito x seja instável. Este algoritmo combina a definição de frequência de termos e a frequência inversa dos documentos para ponderar cada classe y no projeto em questão.

Neste trabalho, a frequência de termos é definida pela quantidade de referências a conceitos da ontologia identificados como impactados (x) e que existem em cada classe do código fonte. A frequência inversa dos documentos avalia quantas classes do código fonte fazem referência aos conceitos impactados.

Para exemplificar, considere que uma classe do código fonte possui referência a 10 conceitos da ontologia, sendo 3 equivalentes ao valor de x. Neste caso, TF (*term-frequency*) representa 3/10 = 0,3. O IDF (*inverse document frequency*) é gerado percorrendo todas as classes da aplicação buscando pelo conceito x. Caso existam 75 classes e o conceito x apareça em 10 dessas classes, então TFIDF é calculado como log(75/10) = 0,87. O valor de TFIDF, neste caso é 0,3 * 0,87 = 0,26.

Os procedimentos para busca de conceitos em cada AST (classe do código fonte) incluem percorrer todas as classes, identificando respectivamente a equivalência dos itens abaixo com os conceitos da ontologia.

- *Atributos*: (1) Nome; e (2) Tipo.
- *Métodos*: (1) Nome; (2) Tipo de retorno; (3) Argumentos, considerando nome ou tipo; e (4) Declarações de variáveis no corpo do método, considerando exceções (bloco try-catch), laços (while, for), condicionais (if) e subblocos, recursivamente.

Diferente da definição do grafo para o algoritmo do PageRank referente a P(y), toda a busca por conceitos em classes do código fonte agora é realizada utilizando o algoritmo de análise de similaridade descrito na Seção 3.2.2, pois não se consegue garantir apenas a equivalência léxica dos conceitos da ontologia com as estruturas do código fonte. Cada referência ao conceito encontrado em cada classe ou interface é considerada para o cálculo de TF. Para ilustrar, considere o código parcial da interface TimecardHome ilustrado pela Figura 3.22.

```
package com.wiley.compBooks.EJwithUML.TimeCardDomain;

import java.util.*;
import java.rmi.*;
import javax.ejb.*;

public interface TimecardHome extends EJBHome
{

  public Timecard create(String employeeId) throws
CreateException,RemoteException;

  public Enumeration findAllForEmployee(String employeeId) throws
FinderException,RemoteException;

  public Enumeration findTimecard(String employeeId, int startDayOfYear, int
startYear) throws FinderException,RemoteException;

  public Timecard findByPrimaryKey(String timecardId) throws FinderException,
RemoteException;
}
```

Figura 3.22 – Código fonte parcial da classe EmployeeBean.

Para o código fonte da Figura 3.22, são identificadas as estruturas de dados: Timecard, employeeId, findAllForEmployee, employeeId, findTimecard, employeeId, Timecard, timecardId, que fazem referência aos conceitos Employee e Timecard. A quantidade de referências ao conceito Timecard neste exemplo é 4 (Timecard, findTimecard, Timecard, timecardId) de uma quantidade total de 8, então TF de Timecard nesta interface é 4/8 = 0,5.

O cálculo de IDF percorre todas as classes da aplicação contabilizando a ocorrência de determinado conceito de forma similar a busca de frequência de termos. No exemplo apresentado, em um universo de 75 classes, o conceito Timecard aparece em 11 classes (ExportFile, RecordTimeWorkflowBean, Test, AddEmployeeWorkflowBean, RecordTimeWorkflow, TimecardHome, TimecardBean, ExportTimeEntriesApplication, Employee, EmployeeBean, TextEntryFrame). O cálculo de IDF é log(75/11) = 0,83.

O valor TFIDF do conceito Timecard para interface TimecardHome é 0,5 * 0,83 = 0,41, o que representa a relação entre a frequência de vezes que o conceito *Timecard* é

identificado na classe *TimecardBean* (TF) e o número total de classes que este conceito é referenciado (IDF).

$$TFIDF(Código = TimecardBean \mid Conceito = Timecard) = 0{,}41$$

A segunda análise foi utilizada para aumentar a acurácia da recomendação. Para tanto, foi aplicada a análise de dependência para calcular a distância conceitual entre as estruturas do código. Esta análise é similar as tradicionais análises estáticas do código fonte, porém considerando agora uma perspectiva semântica.

A Dependência Conceitual é uma métrica proposta por esta tese, resultado da combinação do modelo de Sobreposição Taxonômica (*Taxonomic Overlap*) [Mae02], que avalia a sobreposição de taxonomias definidas por diferentes ontologias, com a análise de dependência, definida por grafos de chamada no código fonte. Essa métrica é utilizada para avaliar a distância entre as diferentes classes da aplicação (dependência) com relação aos conceitos do domínio instáveis.

A análise de Dependência Conceitual (DC) é calculada pela seguinte equação:

$$DC(class_1, conc_1) = \prod_{i=1}^{n-1} \frac{dist(conc_1, conc_{i+1})}{dist(class_1, class_{i+1})}$$

Onde $dist(conc_1, conc_{i+1})$ representa a menor distância entre dois conceitos medida pela quantidade de object properties e $dist(class_1, class_{i+1})$, a menor distância entre duas classes associadas a conceitos diferentes e medida por suas dependências. A dependência de classe é considerada quando uma classe possui uma referência a outra classe do mesmo projeto declarada em seu corpo. Esta análise não considera a distância entre classes pertences ao mesmo conceito e só é empregada quando mais de um conceito da ontologia é considerado instável. Para exemplificar, considere a Figura 3.23.

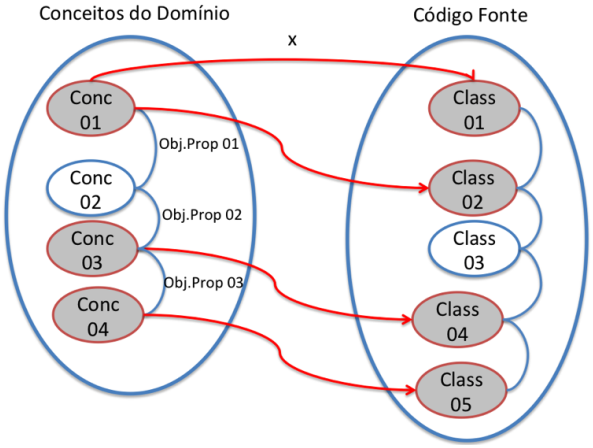


Figura 3.23 – Dependência conceitual.

Supondo que os conceitos Conc01, Conc03 e Conc04 foram identificados como instáveis por uma mudança. Considere também que Conc01 está relacionado a Class01 e Class02, Conc03 está relacionado a Class04 e Conc04 está relacionado a Class05. Neste exemplo, percebe-se que a distância entre Conc01 e Conc03 é 2; Conc01 e Conc04 é 3; Class01 e Class04 é 3; e Class01 e Class05 é 4.

A distância conceitual da Class01 considerando Conc01 pode ser calculada pelas suas distâncias relativas do conceito base (Conc01) com os demais impactados, bem como da classe base (Class01) com as demais rastreadas, conforme:

$$DC(Class01, Conc01) = \frac{dist(Conc01, Conc03)}{dist(Class01, Class04)} \times \frac{dist(Conc01, Conc04)}{dist(Class01, Class05)} = \frac{2}{3} \times \frac{3}{4} = 0{,}5$$

Para calcular DC, o modelo primeiro avalia se existe mais de um conceito impactado. Se sim, percorre-se cada par de conceitos impactados e recupera-se suas respectivas classes do código fonte. O cálculo da distância entre conceitos é realizado encontrando o menor caminho entre o conceito base (x) e os demais conceitos impactados considerando todos os objects properties. A análise de dependência do código fonte é realizada identificando se na classe base (y) existe referência a uma classe rastreada ao segundo conceito. Caso não encontre, a mesma busca é realizada em todos os tipos encontrados na segunda classe (segundo nível) e assim por diante, recursivamente. A identificação dos tipos nas classes considera:

- *Atributos*: (1) Tipo; e (2) Tipo declarado em subbloco do tipo (declaração estática).
- *Métodos*: (1) Tipo de retorno; (2) Tipo dos argumentos; e (3) Declarações de variáveis no corpo do método, considerando exceções (bloco try-catch), laços (while, for), condicionais (if) e subblocos, recursivamente.

Para ilustrar, considere os seguintes conceitos instáveis: Timecard e RecordTime. As seguintes estruturas foram identificadas como impactadas:

- *Conc01:* Timecard
    - *Classes*: TimecardHome, TimecardBean, Timecard.
- *Conc2:* RecordTime
    - *Classes*: RecordTimeWorkflowBean, RecordTimeWorkflowHome, RecordTimeWorkflow, RecordTimeServlet, ExportTimeEntriesApplication.

A distância conceitual de Employee e RecordTime é 1 devido ao relacionamento dos conceitos pela object property *hasCurrent* (Timecard hasCurrent RecordTime). O próximo passo é calcular a menor distância entre todas as classes associadas a Conc01 e Conc02. O resultado é apresentado na Tabela 3.6.

Tabela 3.6 –Dependência conceitual.

Conceito: RecordTime → Timecard			**Conceito: Timecard → RecordTime**		
Classe Origem	**Classe Destino**	**Dist.**	**Classe Origem**	**Classe Destino**	**Dist.**
RecordTime	Timecard	1	Timecard	RecordTime	1
	TimecardHome	-		RecordTimeWorkflowBean	1
	TimecardBean	-		RecordTimeWorkflowHome	-
RecordTimeWorkflowBean	Timecard	-		RecordTimeWorkflow	-
	TimecardHome	-		RecordTimeServlet	-
	TimecardBean	-	TimecardHome	ExportTimeEntriesApplication	1
RecordTimeWorkflowHome	Timecard	-		RecordTimeWorkflowBean	-
	TimecardHome	-		RecordTimeWorkflowHome	-
	TimecardBean	-		RecordTimeWorkflow	-
RecordTimeWorkflow	Timecard	-		RecordTimeServlet	-
	TimecardHome	-	TimecardBean	ExportTimeEntriesApplication	-
	TimecardBean	-		RecordTimeWorkflowBean	1
RecordTimeServlet	Timecard	-		RecordTimeWorkflowHome	-
	TimecardHome	-		RecordTimeWorkflow	-
	TimecardBean	-		RecordTimeServlet	-
ExportTimeEntriesApplication	Timecard	-	Timecard	ExportTimeEntriesApplication	1

Caso não seja encontrada a dependência conceitual no código fonte, optou-se por atribuir um redutor para 10% na distância para o cálculo da probabilidade condicional. Este valor foi obtido através de uma calibragem empírica com o objetivo de aumentar a probabilidade de determinada classe caso a equivalência seja detectada. O resultado final pode ser observado na Tabela 3.7.

Tabela 3.7 – PageRank para os conceitos da ontologia.

Classe	Distância
RecordTimeWorkflowBean	0.1
RecordTimeWorkflowHome	0.1
RecordTimeWorkflow	0.1
Timecard	1.0
TimecardBean	0.1
TimecardHome	1.0
ExportTimeEntriesApplication	0.1
RecordTimeServlet	0.1

Conforme apresentado, a probabilidade de $P(y|x)$ é dada pela média geométrica do cálculo TFIDF e DC:

$$P(y|x) = \sqrt{TFIDF(y|x) \times DC(y|x)}$$

Conforme o exemplo apresentado:

$$TFIDF(Código = TimecardBean \mid Conceito = Timecard) = 0{,}41$$

$$DC(Código = TimecardBean \mid Conceito = Timecard) = 0{,}1$$

Logo:

$$P(Código = TimecardBean|Conceito = Timecard) = \sqrt{0{,}41 \times 0{,}1} = 0{,}2$$

Esta medida é calculada para todas as classes do código fonte dada a definição de um conceito como instável, isto é, que suas estruturas serão possivelmente impactadas pelo requisito de mudança.

Foi identificado que a distribuição da amostra pode possuir uma dispersão significativa dos valores, gerando um conjunto de artefatos com baixa probabilidade de impacto. Para eliminar a elementos com baixa relevância, foram excluídos àqueles abaixo do desvio padrão.

Revendo o cenário motivacional, a solicitação de mudança apresentada na Figura 3.5 inclui os seguintes conceitos e propriedades da ontologia: *submit, record, time, hours*. De posse destes conceitos e com a ontologia e código fonte informados, o sistema:

1. carrega a ontologia e o código fonte;
2. realiza automaticamente a população da ontologia e rastreabilidade;
3. realiza a classificação dos modelos usando PageRank;
4. identifica classes e propriedades na ontologia instáveis;
5. calcula TF, IDF e DC de cada classe do código fonte; e
6. apresenta as estruturas do código fonte possivelmente impactadas (instáveis), incluindo referências a classes, atributos e métodos.

O resultado resumido da execução pode ser visto na Figura 3.24. Conforme apresentado no cenário motivacional, a inspeção do código fonte apontou que, para o requisito de mudança, seria necessário alterar as classes RecordTimeWorkflowBean e suas interfaces RecordTimeWorkflow e RecordTimeWorkflowHome, bem como na classe

de fronteira RecordTimeServlet para validação da data. Todas estas classes foram identificadas como relevantes pelo modelo apresentado.

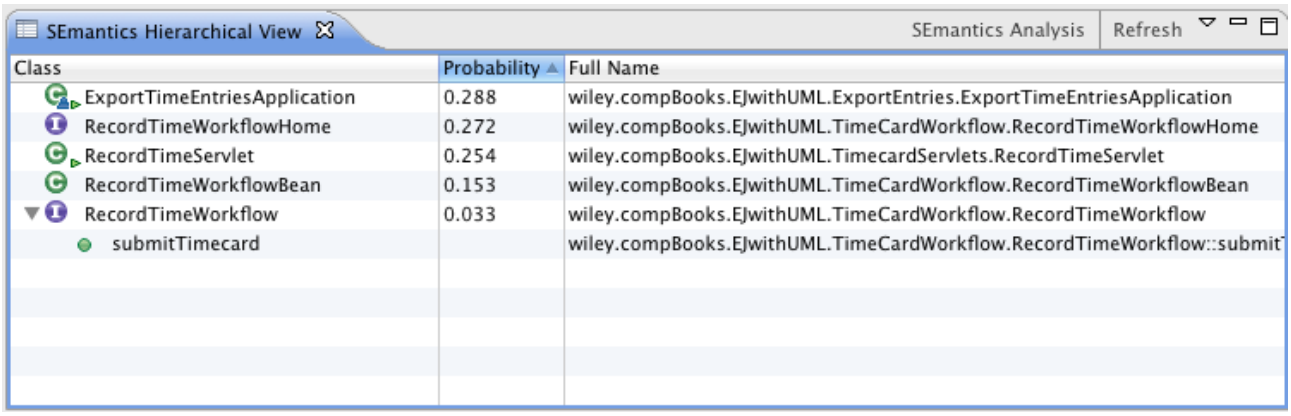


Figura 3.24 – Resultado da análise de impacto para o cenário motivacional.

Observa-se também que a classe ExportTimeEntriesApplication foi definida como relevante, o que reforça a necessidade de inspeção do resultado por um especialista. O motivo de sua relevância é a grande frequência que os conceitos instáveis foram identificados em seu construto.

Ainda assim, este modelo de análise de impacto proporciona uma redução das classes inspecionadas de 75 para 5, o que equivale analisar apenas 6,67% da quantidade total classes do sistema, com uma taxa de erro, neste exemplo, de 1,33% do escopo total.

3.4. Considerações sobre o Capítulo

A análise de impacto em software é uma tarefa de estimar as partes do software que serão impactadas quando uma mudança proposta é realizada. A informação de análise de impacto pode ser utilizada em diferentes momentos durante o desenvolvimento de software como o planejamento, execução e acompanhamento de mudanças, ou mesmo a análise de efeitos colaterais (no caso de testes de regressão). Uma forma tradicional de estimar impacto é usando informações estáticas do código fonte (isto é, suas dependências) para identificar as partes do software que podem ser afetadas por determinadas mudanças. Por exemplo, inserindo mudanças no código e buscando as suas dependências usando grafos de chamada. Estas técnicas tradicionais são geralmente imprecisas e tendem a superestimar os efeitos de uma mudança [Jon07].

O modelo de análise de impacto proposto combina uma série de abordagens e técnicas visando melhorar a acurácia na recuperação de estruturas relevantes frente a um

requisito de mudança. Esta combinação foi resultado de uma série de análises empíricas sobre o objeto de estudo. Para tanto, foram definidos dos submodelos: Modelo de Rastreabilidade e Modelo Probabilístico. O primeiro visa rastrear conceitos do domínio com estruturas do código fonte e o segundo calcular a probabilidade de mudança frente a um requisito. De forma geral, este modelo necessita de três informações para realizar a análise de impacto e cálculo da influência dos conceitos: (1) o projeto da aplicação e seu código fonte; (2) a ontologia do domínio; e (3) um texto livre com a requisição de mudança (RdM).

O modelo de análise de impacto realiza então a carga do código fonte da aplicação, organizados em AST e traduzidos em uma ontologia OWL, e a ontologia do domínio, organizado em um modelo OWL.

Após a carga, é instanciado o Modelo de Rastreabilidade que percorre todas as estruturas do código fonte, populando a ontologia de domínio com referências a instâncias da ontologia do código fonte. Este mapeamento percorre todas as classes do código fonte, analisando seus atributos e métodos, bem como suas estruturas aninhadas. Este procedimento realiza a busca de referências a conceitos do domínio no código fonte utilizando um analisador de similaridade léxico e semântico. Este analisador realiza a categorização, normalização e comparação de cada um dos termos de busca, fazendo uso de PLN. O resultado deste modelo é uma ontologia de domínio rastreada a estruturas do código fonte.

Com base no RdM e no Modelo de Rastreabilidade, o Modelo de Probabilidade é instanciado. Este modelo identifica inicialmente no RdM os conceitos na ontologia do domínio através do mesmo analisador de similaridade léxico e semântico descrito. Para o cálculo da probabilidade, foi utilizado o modelo de recuperação de informação RCB que é organizado em probabilidade independente e condicional. Para o calculo de probabilidade independente dos conceitos do domínio e código fonte, foi utilizado o algoritmo de classificação PageRank com uma configuração semelhante ao motor de busca do Google. Para a probabilidade condicional, foi utilizada uma análise combinada de TFIDF, para análise de frequência dos conceitos em cada código fonte, e DC, para identificação da distância conceitual através da sobreposição de grafos de chamadas do código fonte com a organização estrutural dos conceitos da ontologia. Como resultado deste modelo, é apresentado um cálculo de probabilidade associado a cada classe do código fonte, indicando a relevância de cada estrutura do código, incluindo atributos e métodos que potencialmente serão impactados pelo requisito de mudança.

4. SEMANTICS: UMA FERRAMENTA PARA ANÁLISE DE IMPACTO

Este capítulo detalha a arquitetura da ferramenta SEmantics (*Semantic Software Engineering*) desenvolvida para verificar a viabilidade do Modelo de Análise de Impacto sugerido. Este detalhamento será estruturado conforme os Modelos de Rastreabilidade e Probabilístico apresentados no Capítulo 3, visando relacionar a parte conceitual com sua implementação.

A implementação da ferramenta SEmantics foi realizada como uma extensão (*plugin*) do IDE Eclipse [Ecl12] utilizando a linguagem de programação Java [Jav12]. O propósito de estender uma IDE (*Integrated Development Environment*) foi integrar o modelo de análise de impacto proposto com uma ferramenta tradicional de desenvolvimento e manutenção do código fonte.

Antes de decidir por qual IDE, foi necessário realizar um estudo de viabilidade técnica que incluiu analisar APIs de manipulação de código fonte, ontologias, recuperação de informação e processamento de linguagem natural. Os resultados desses estudos foram inicialmente publicados em [Nol08] e posteriormente descritos em [Nol11].

Quanto ao escopo de implementação, neste primeiro momento a ferramenta apresenta suporte apenas para aplicações que utilizam a linguagem de programação Java. Esta restrição é devido à dependência das bibliotecas utilizadas para o *parser* do código fonte.

Durante a implementação da ferramenta, foram observados aspectos para permitir a extensão dos modelos relacionados a diferentes produtos de trabalho. O modelo de rastreabilidade foi estruturado para que fosse flexível o suficiente para processar documentos UML (XMI - *XML Metadata Interchange*), casos de uso e de teste descritos em linguagem natural, entre outros, expandindo o escopo de código fonte desenvolvido neste trabalho. Foram realizados testes de viabilidade com diferentes produtos de trabalho conforme descritos em [Nol08], porém a primeira versão da ferramenta SEmantics manteve apenas elos de rastreabilidade entre ontologia e código fonte.

Por último, acredita-se que, apesar das diversas otimizações realizadas, ainda serão necessários aperfeiçoamentos com relação a performance e desempenho. Para a primeira versão da ferramenta, é improvável que se consiga chegar em uma arquitetura

ideal em termos de flexibilidade e desempenho. Estas e outras limitações serão melhor discutidas no Capítulo 6, que apresenta as considerações finais e os trabalhos futuros.

4.1. Arquitetura SEmantics

A ferramenta SEmantics foi definida como uma arquitetura de referência visando possíveis extensões. O objetivo é estabelecer um vocabulário comum para servir como base de discussão em futuras implementações. Para tanto, a descrição da arquitetura SEmantics concentra-se principalmente na definição das responsabilidades dos elementos que a compõe, suas APIs e aspectos estruturais e comportamentais.

Esta seção apresenta uma visão geral da arquitetura desenvolvida através de diferentes perspectivas da aplicação. Será apresentada uma visão estrutural, que descreve a infraestrutura básica e bibliotecas, e comportamental, que descreve os principais aspectos dinâmicos associados a atividade de análise de impacto.

4.1.1. Visão Estrutural da Aplicação SEmantics

Para o desenvolvimento, foram utilizadas as seguintes bibliotecas e APIs:

1. *JDT (Java Development Tools)* [Jdt12]: é um projeto que implementa buscas precisas por declarações, referências a pacotes, tipos, métodos e campos no escopo do *workspace* de um projeto Java. Os complementos JDT fornecem as seguintes APIs:
    a. APT: para suporte de anotações em projetos Java 5;
    b. Debug: para suporte a debug em conformidade com JPDA (*Java Platform Debugger Architecture*);
    c. Text: que fornece editores Java com uma série de funcionalidades, como cores conforme sintaxe e palavras reservadas, formatação de JavaDoc, assistente e seleção de código, etc.;
    d. UI: que fornece contribuições do *Package Explorer*, visão hierárquica de código, *outline*, etc.; e
    e. Core: utilizada para construção de código Java e navegação em sua árvore AST, como fragmentos de pacotes, unidades de compilação,

classes, tipos, métodos e atributos, além de indexação para busca e computação de tipos hierárquicos;

2. *Jena* [Jen12]: é um projeto cujo objetivo é proporcionar um *framework* Java para construção de aplicações para Web Semântica. Jena fornece uma coleção de ferramentas e bibliotecas Java de apoio ao desenvolvimento de aplicações que inclui:
    a. uma API para leitura, processamento e escrita de dados RDF em XML, N-tripples e formato Turtle;
    b. uma API para manipulação de ontologias OWL e RDF;
    c. um motor de inferência baseado em regras para raciocínio utilizando RDF ou OWL como origem de dados;
    d. armazenamento em disco de grande quantidade de triplas RDF;
    e. um motor de busca em conformidade com SPARQL;
    f. servidores que permitem a publicação de dados RDF em outras aplicações, utilizando uma variedade de protocolos;
3. *JUNG (Java Universal Network/Graph Framework)* [Jun12]: é um *framework* de código livre Java cujo objetivo é fornecer uma linguagem comum e extensível para modelagem, análise e visualização de dados que podem ser representados como grafos ou rede. A arquitetura JUNG suporta uma variedade de representações de entidades e seus relacionamentos, facilitando a criação de ferramentas analíticas para complexos conjuntos de dados. Estas análises examinam as relações entre as entidades, assim como os metadados anexados a cada entidade e relação. Este framework inclui a implementação de um grande número de algoritmos da teoria dos grafos, mineração de dados, análise de rede social, tais como rotinas para clusterização, decomposição, otimização e cálculo de distância de rede, fluxos e outras medidas importantes (como centralidade, PageRank, etc.);
4. *JWNL (Java WordNet Library)* [Jwn12]: é uma API Java que viabiliza o acesso ao dicionário relacional WordNet (versões 2.0 e 3.0), fornecendo meios para a descoberta de relações (sinonímia, antonímia, etc.) e processamento morfológico;

5. Maui-Indexer (*Multi-purpose automatic topic indexing*) [Mau12]: é uma API que identifica tópicos em documentos dependendo de uma tarefa, palavra reservada ou índices. Esta API fornece algoritmos de *stemmer* e identificação de *stopwords* em inglês, francês e espanhol.

As APIs descritas acima foram utilizadas respectivamente para:

1. acessar e manter o código fonte da aplicação, realizando *parser* do documento, sua conversão para AST e navegação em sua estrutura;
2. criar e manipular ontologias OWL;
3. utilizar algoritmos baseado na teoria de grafos, incluindo PageRank;
4. acessar a WordNet para recuperação de relações conceituais entre termos; e
5. comparação léxica utilizando o algoritmo de stemmer.

Foi definida uma arquitetura centralizada para a ferramenta SEmantics utilizando um modelo clássico baseado em camadas. Esta arquitetura é chamada monolítica, pois é estruturada em camadas lógicas que obrigatoriamente residem no mesmo nó físico, isto é, a IDE onde o *plugin* está sendo executado, e seus componentes não podem ser distribuídos. A arquitetura da aplicação é organizada em camadas lógicas conforme o diagrama de pacotes ilustrado na Figura 4.1.

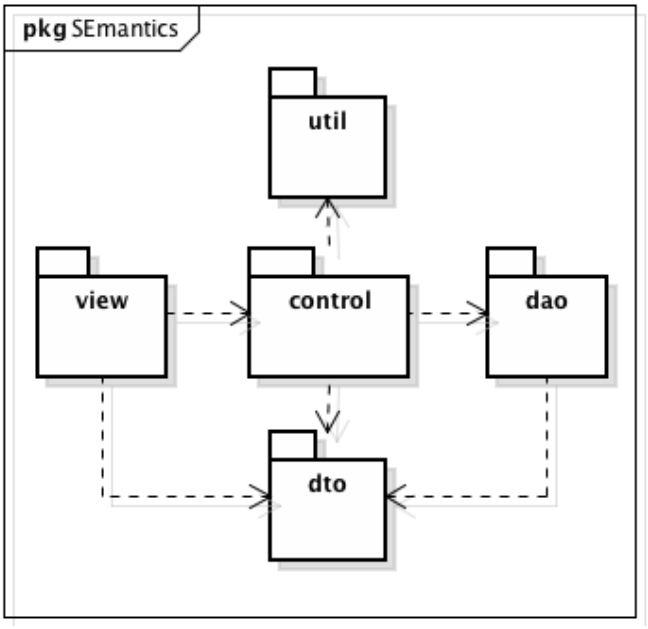


Figura 4.1 – Diagrama de Pacotes da ferramenta SEmantics.

A Figura 4.1 apresenta a visão lógica da arquitetura, ilustrando a dependência entre os pacotes em nível de camadas. Entre as camadas da aplicação, destacam-se três

equivalentes ao padrão de projeto MVC (*Model-View-Controller),* identificadas pelos pacotes *view*, *control* e *dao*. Estas camadas são responsáveis por tratar questões relativas a interface com o usuário (*view*), via *plugin* do Eclipse ou linha de comando, regras para análise de impacto (*control)*, incluindo os controladores de rastreabilidade e probabilidade, e persistência (*DAO – Data Access Object),* responsável pelo acesso e *parser* do código fonte e ontologia a partir do sistema físico de arquivos.

4.1.1.1. Camada de Visão.

A camada de visão é definida pelo pacote *view* e ilustrada pela Figura 4.2. Esta camada é responsável pelo acesso a funcionalidade de análise de impacto. Duas classes principais são definidas nesta visão: `SEmanticsView`, responsável pelo acesso a funcionalidade de análise de impacto através de linha de comando, e `SEmanticsPlugin`, invocada pela IDE para carregar os painéis de comunicação com o usuário.

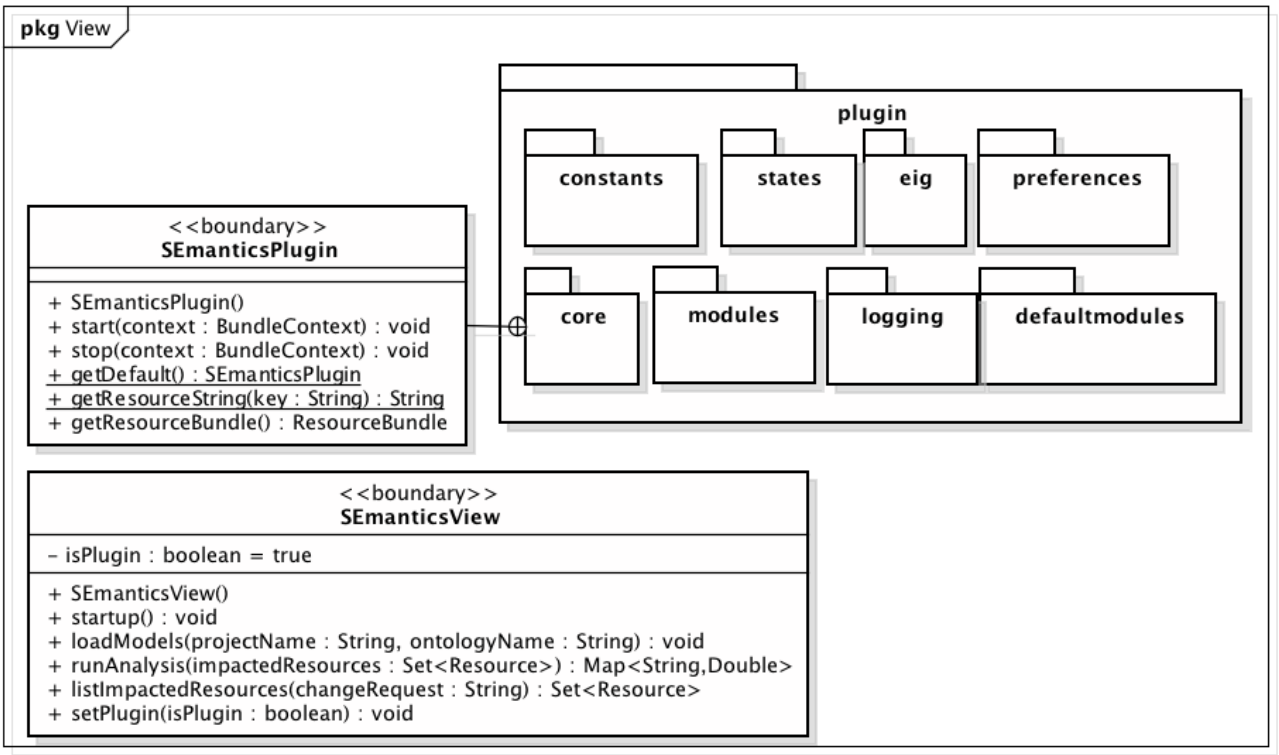


Figura 4.2 – Diagrama de classes do pacote *view*.

A classe `SEmanticsView` viabiliza o acesso a funcionalidade de forma direta, através dos métodos: `startup`, responsável por inicializar contextos e controladores; `loadModels`, utilizado para carregar os modelos, fornecendo a referência física de onde se encontram a ontologia e o código fonte do projeto Java; e `runAnalysis`, responsável

por receber um requisito de mudança e executar o cálculo de análise de impacto, retornando uma lista de classes ponderadas conforme sua relevância.

A classe `SEmanticsPlugin` é outra alternativa de acesso a aplicação SEmantics através de um *plugin* do Eclipse. Ela é responsável por inicializar os contextos e controles de painéis com base no arquivo de configuração *plugin.xml.* O arquivo de configuração do *plugin* define as classes responsáveis para a entrada e saída de dados referentes a análise de impacto. Este painéis incluem uma nova entrada no menu do Eclipse e ícone no painel *default* para acesso à funcionalidade, bem como um painel na parte inferior com os resultados da execução. A Seção 4.2 apresentará o uso da ferramenta, onde cada componente será explicado em detalhes.

4.1.1.2. Camada de Controle.

A camada de controle é definida pelo pacote *control* e ilustrada pela Figura 4.3. Esta camada é responsável pelo acesso a funcionalidade de análise de impacto e possui quatro classes principais: `SEmanticsController`, `TraceabilityModelController`, `ProbabilityModelController` e `SimilarityController`.

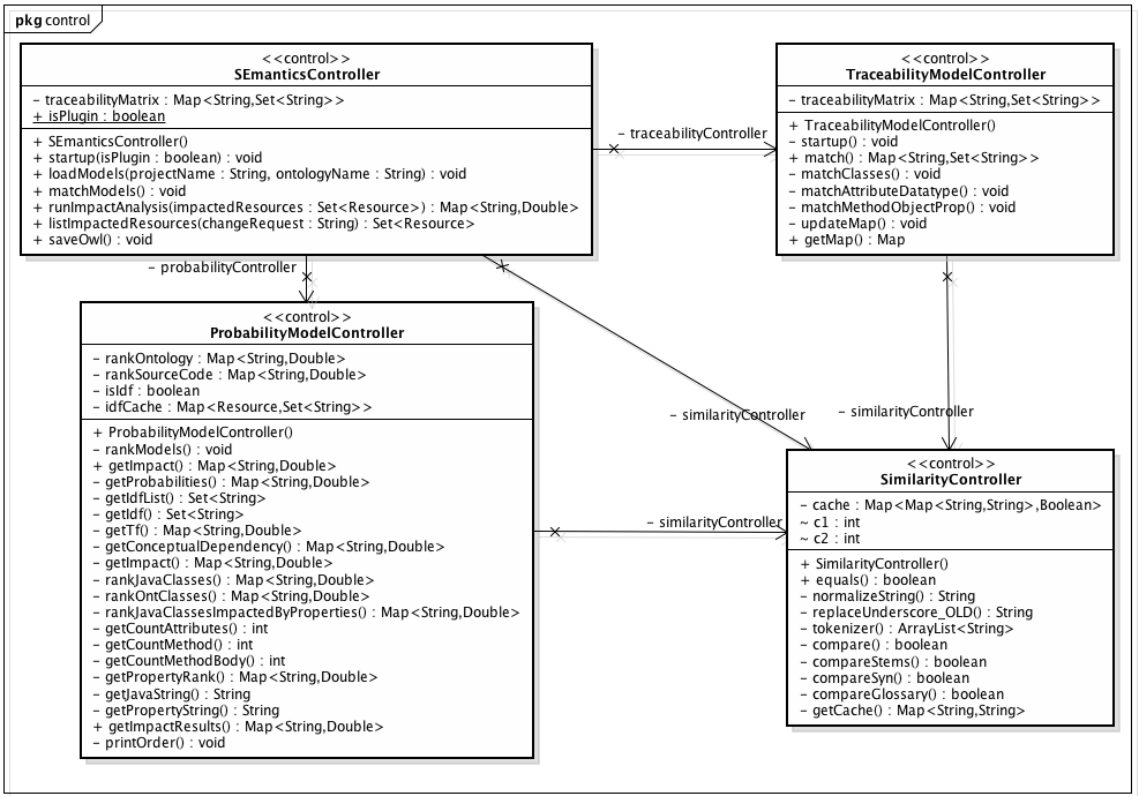


Figura 4.3 – Diagrama de classes do pacote *control*.

A classe `SEmanticsController` é a classe base da camada de controle. Ela é responsável por instanciar os modelos de rastreabilidade (`TraceabilityModelController`) e probabilidade (`ProbabilityModelController`), bem como a análise de similaridade (`SimilarityController`) utilizada por estes dois modelos. A classe `SEmanticsController` recebe da camada de visão as referências aos arquivos do projeto Java e ontologia e utiliza suas classes de persistência (*DAO*) para recuperação dos modelos. Estes modelos são convertidos pelos *DAOs* em objetos DTOs (*Data Transfer Object* - objetos de transferência de dados) para definição da matriz de rastreabilidade e cálculo de relevância.

A matriz de rastreabilidade é de responsabilidade da classe `TraceabilityModelController`, que mantém a referência a JWNL para acesso a WordNet. Este acesso permite a categorização de palavras, identificando substantivos, e recuperação de sinonímia entre termos. Ela também recebe o controlador `SimilarityController` responsável pela análise léxica e semântica entre termos, que será descrito em detalhes em seguida.

Dentre os principais métodos da classe `TraceabilityModelController`, destaca-se o *match*, responsável pelo mapeamento do código fonte com a ontologia. Este método percorre cada unidade de compilação do código fonte, identificando classes, atributos e métodos para realizar sua rastreabilidade com classes, datatype properties e object properties da ontologia, respectivamente. O objetivo da classe `TraceabilityModelController` é popular a ontologia de domínio com as referências as estruturas de baixo nível do código fonte (atributo, método e classe).

A classe `ProbabilityModelController` é responsável pela identificação da relevância dos elos de rastreabilidade da ontologia de rastreabilidade. Ela possui o método `getImpact` responsável pela recuperação do impacto, que inclui: (1) definir o cálculo da probabilidade marginal P(x) de cada classe na ontologia utilizando o algoritmo PageRank; (2) definir a probabilidade marginal P(y) de cada classe do código fonte utilizando o algoritmo PageRank; e (3) definir a probabilidade condicional P(y|x) pela classificação de cada aresta do nodo, utilizando a análise TFIDF e DC. Ao final, o cálculo de probabilidade do modelo RCB é realizado com base nas probabilidades previamente identificadas.

4.1.1.3. Camada de Persistência e Dados.

A Figura 4.4 ilustra a relação das camadas de persistência (definida pelo pacote *dao*) e de dados (definida pelo pacote *dto*). As classes DTO (*Data Transfer Object*) são referenciadas em todas as camadas, conforme Figura 4.1, porém optou-se por apresentar as relações estruturais associadas às classes DAO para ilustrar sua instanciação. O pacote *dao* inclui as classes `OntModelDAO`, `JavaProjectDAO` e `JavaAstDAO`. O pacote *dto* possui as classes `OntModelDTO`, `OntClassDTO`, `JavaAstDTO`, `JavaClassDTO` e `JavaProjectDTO`.

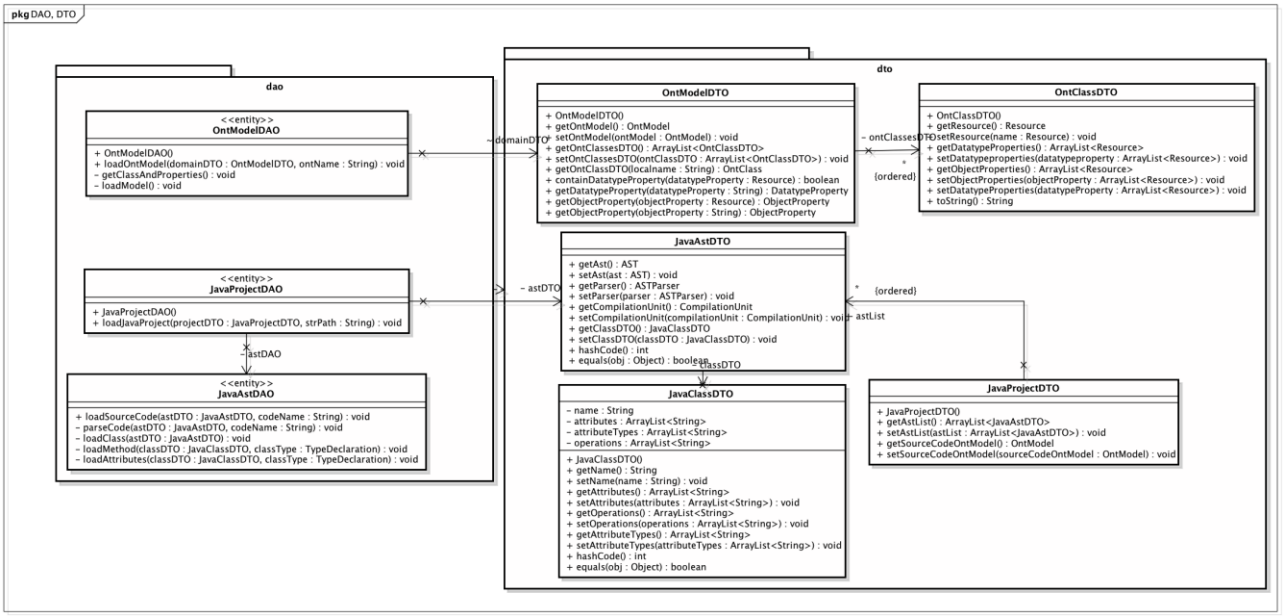


Figura 4.4 – Diagrama de classes de persistência.

A classe `OntModelDAO` recupera e realiza o *parser* da ontologia informada pela camada de visão, criando uma instância do `OntModelDTO`. Essa ontologia é carregada utilizando a classe `OntModel` da API Jena e suas classes, datatype e object properties são identificadas para facilitar posteriores buscas e comparações. O processo de carga da ontologia será melhor descrito pela visão comportamental.

A classe `JavaProjectDAO` é responsável por fazer a carga e *parser* dos arquivos do código fonte. Para tanto, a mesma recupera todos os arquivos com extensão *java* a partir do caminho informado pela camada de visão, incluindo subdiretórios. Os arquivos são armazenados em uma lista de `File` e recuperados por uma classe utilitária chamada `FileListing`. Em seguida, a classe `JavaAstDAO` realiza o *parser* de cada arquivo em uma unidade de compilação AST utilizando a API JDT, carregando em uma lista de `JavaAstDTO`. Esse DTO inclui todas as compilações de classes do código fonte,

incluindo classe, atributos e métodos. Uma vez carregada todas as classes do projeto Java, é gerada a ontologia do código fonte com base nos atributos e métodos de cada classe.

4.1.2. Visão Comportamental da Aplicação SEmantics

Uma visão comportamental de alto nível da aplicação SEmantics pode ser ilustrada pelo diagrama de atividades da Figura 4.5.

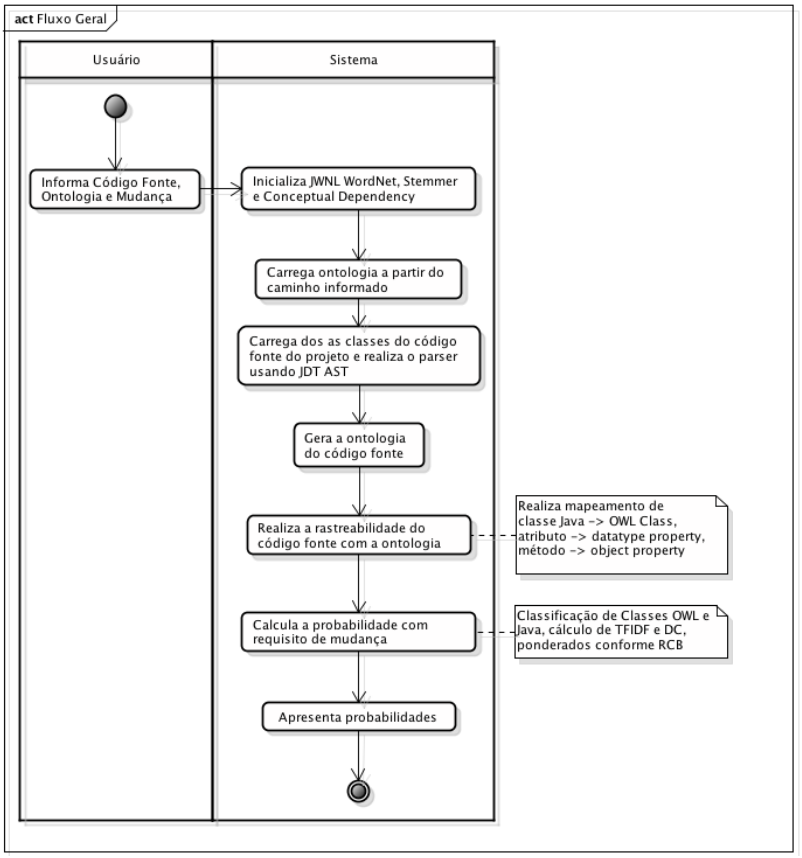


Figura 4.5 – Diagrama de atividades do fluxo geral.

O fluxo de processo para a análise de impacto inicia quando o usuário informa o endereço do código fonte e da ontologia, bem como um texto livre com a requisição de mudança. Neste momento, a aplicação é instanciada identificando qual classe de visão iniciou o processo (linha de comando ou *plugin*) para carga adequada dos painéis de interface. A configuração inicial inclui a instanciação dos DAOs `OntModelDAO` e `JavaProjectDAO` para as respectivas cargas dos modelos nos DTOs `OntModelDTO` e `JavaProjectDTO`. Esta configuração inicial também instancia os controles de rastreabilidade (`TraceabilityModelController`), probabilidade (`ProbabilityModelController`) e similaridade (`SimilarityController`), bem

como as classes utilitárias `ConceptualDependency`, `RankOntology` e `RankSourceCode`.

A carga da ontologia inclui a leitura e *parser* do arquivo OWL em um objeto `OntModel` da API Jena e sua atribuição em um objeto `OntModelDTO`, que categoriza a priori classes e propriedades OWL para otimizar buscas em comparações posteriores. Esta carga também inclui os mecanismos necessários para geração dos elos de rastreabilidade utilizando a classe OWL *Associacao*.

A carga das classes do código fonte ocorre percorrendo a estrutura de diretório informada e identificando arquivos com extensão Java. Em seguida, a classe `JavaAstDAO` realiza o rastreamento de cada arquivo identificado, gerando uma AST correspondente para ser armazenada em um objeto `JavaAstDTO`. Este objeto também categoriza o nome da classe, seus atributos e métodos para otimizar as buscas em comparações posteriores. Cada elemento é recuperado percorrendo a árvore do código e identificando o nome do método e o nome e tipo dos atributos. Por fim, a ontologia do código fonte é gerada para fins de rastreabilidade. A Figura 4.6 apresenta o diagrama de sequência que ilustra o processo descrito.

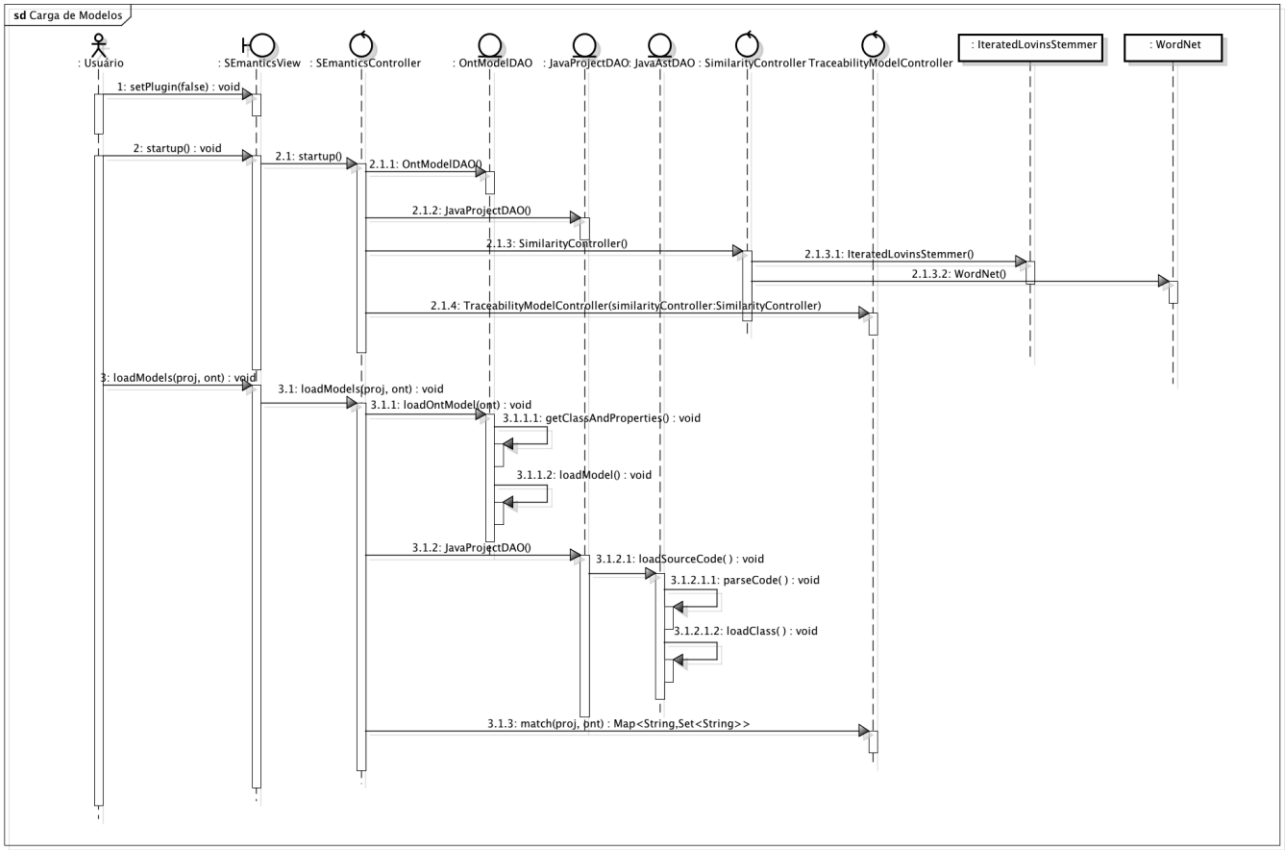


Figura 4.6 – Diagrama de sequência de carga de modelos.

Após a carga da ontologia e código fonte, ocorre a definição da matriz de rastreabilidade, populando a ontologia de domínio com associações às estruturas de código fonte. Este mapeamento ocorre conforme ilustrado pelo diagrama de sequência da Figura 4.7.

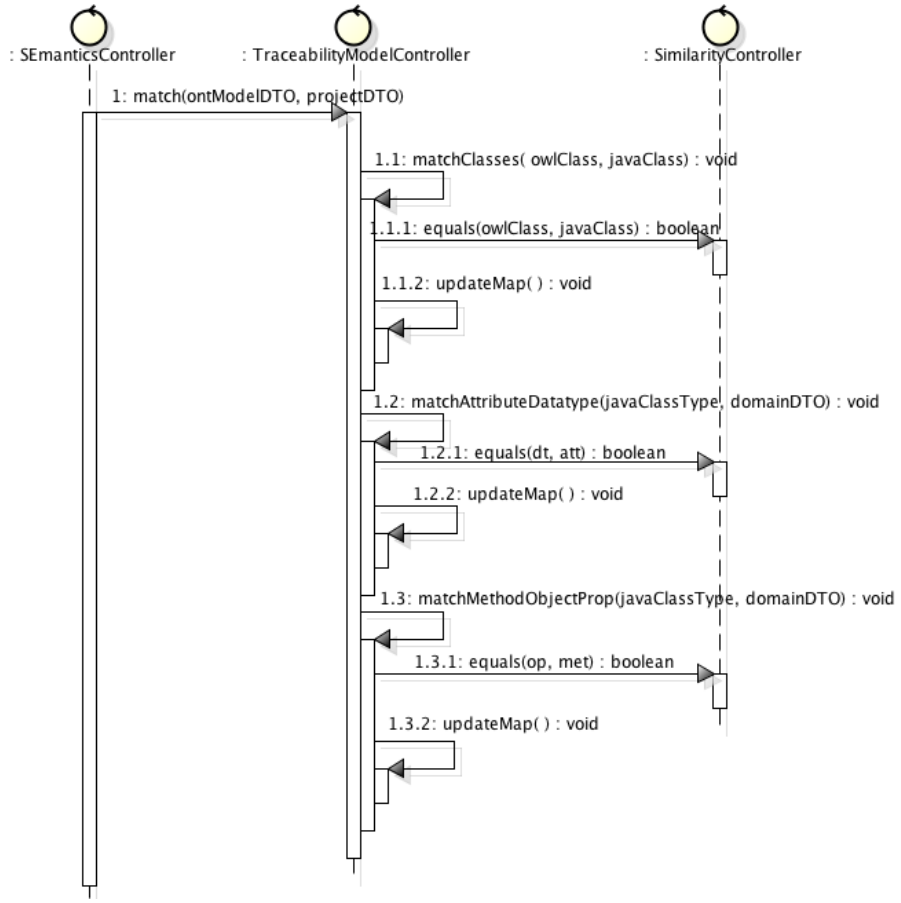


Figura 4.7 – Diagrama de sequência para o modelo de rastreabilidade.

A Figura 4.7 ilustra o processo executado para cada classe do código fonte, no qual é realizada uma análise de similaridade com seus membros, considerando se existe equivalência da classe Java com alguma Classe OWL, de um atributo Java com algum datatype property OWL ou método Java com object property OWL. Para cada similaridade identificada, é gerado ou atualizado um elo de rastreabilidade na ontologia de domínio.

Toda a comparação entre estruturas de código e recursos da ontologia, seja no modelo de rastreabilidade ou dependência, é realizada utilizando a classe `SimilarityController`, conforme ilustrado pela Figura 4.8. A análise ocorre entre duas strings, ignorando `getters` e `setters`, normalizando e gerando *tokens* de todas as strings para posterior comparação. As comparações consideram uma perspectiva léxica pelo *stem* dos termos e uma perspectiva semântica, categorizando elementos como substantivos e recuperando seus sinônimos da WordNet e glossário da aplicação.

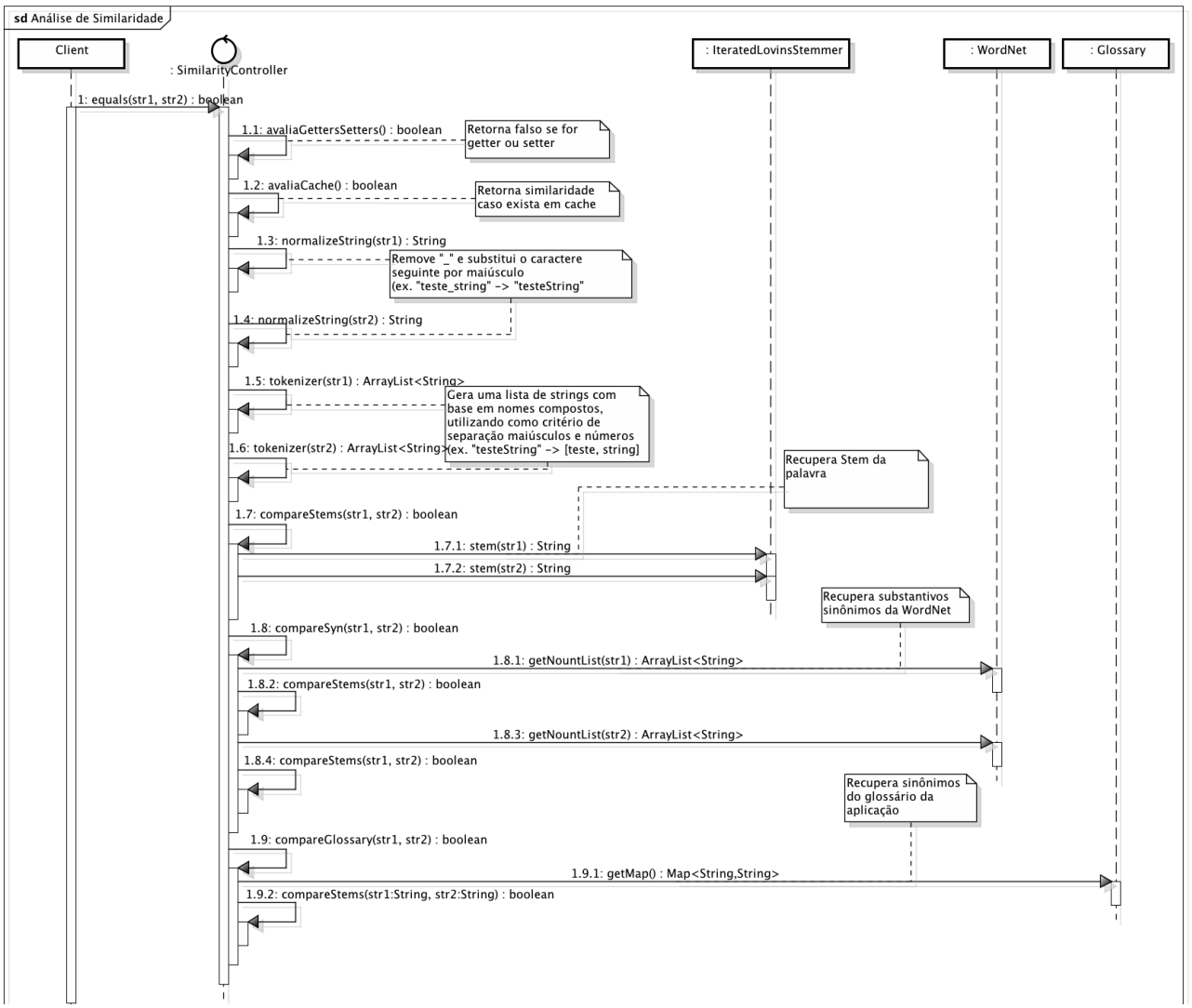


Figura 4.8 – Diagrama de sequência para análise de similaridade.

A classe `SimilarityController` é responsável por receber duas strings e verificar se são similares. Esta análise inclui:

1. Ignorar métodos `getters` e `setters`;
2. Avaliar o *cache* das comparações previamente executadas para otimizar o processo evitando redundância;
3. Normalizar cada string, removendo o caractere *underscore* e substituindo a primeira letra subsequente a esse caractere por caixa alta. Por exemplo, “esta_string_exemplifica” será transformada em “estaStringExemplifica”;
4. Quebrar todas as strings compostas (que o segundo termo inicia em maiúsculo ou números) em uma lista de termos passíveis de comparação. Por exemplo, “estaStringExemplifica1valor” será transformada em uma lista [esta, string, exemplifica, valor];

5. Para cada elemento das duas listas (associadas aos dois termos da comparação), (a) comparar os *stems* dos termos, (b) os *stems* dos sinônimos dos termos substantivos recuperados da WordNet e (c) os *stems* dos termos recuperados do glossário da aplicação.

Após a geração da matriz de rastreabilidade, a classe `ProbabilityModelController` avalia a relevância de cada elo conforme Figura 4.9.

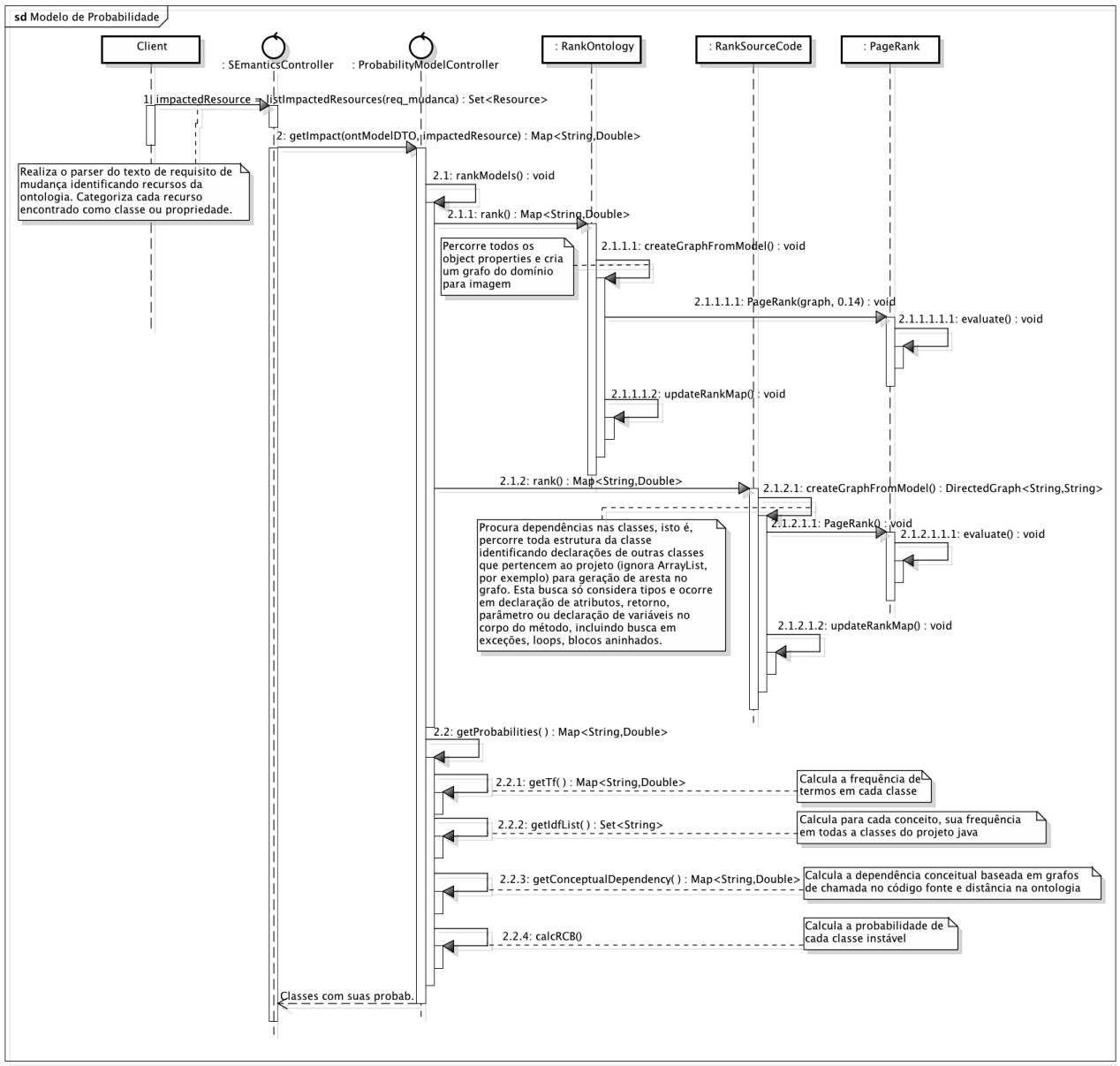


Figura 4.9 – Diagrama de sequência para o modelo de probabilidade.

A classe `SEmanticsController` recebe como argumento a matriz de rastreabilidade e um texto livre com o requisito de mudança. A partir deste texto, o sistema identifica as classes e propriedades da ontologia consideradas instáveis e propensas a mudanças. Com a matriz de rastreabilidade e recursos da ontologia impactados, o sistema inicia o cálculo de probabilidade.

Inicialmente é realizada a classificação utilizando o algoritmo PageRank de ambos os modelos: ontologia e código fonte. O grafo da ontologia é gerado com base em seus object properties. O grafo do código fonte é realizado percorrendo cada classe do projeto e verificando suas dependências com outras classes do mesmo projeto. Esta análise ignora classes definidas em bibliotecas como, por exemplo, “`ArrayList`”, “`Set`”, etc. A análise de dependência é realizada em:

- atributos: identificando seus tipos;
- métodos: identificando recursivamente tipos de retorno, argumentos ou os tipos de variáveis declaradas em seu corpo, incluindo uma análise em laços, blocos de exceção e sub-blocos.

Logo em seguida, é realizado o cálculo de frequência (TF) que um conceito aparece em uma classe utilizando o controle `ProbabilityModelController` e `SimilarityController`, comparando a quantidade total de conceitos que existem nesta mesma classe. Após é identificado quais os conceitos que aparecem em cada classe do projeto Java (IDF). A Figura 4.10 ilustra o fluxo para o cálculo da probabilidade condicional TFIDF.

Após o cálculo de TFIDF, a dependência conceitual (DC) é calculada com base no grafo de chamada do código fonte e na distância entre dois conceitos na ontologia conforme descrito pela Figura 4.11.

A distância conceitual avalia a equivalência entre dois conceitos na ontologia relacionados por object properties, comparado a distância entre as classes do código fonte associadas a estes conceitos. O resultado, juntamente com o valor de TFIDF, é utilizado para o cálculo da probabilidade condicional de cada classe no código fonte impactada por um conjunto de recursos da ontologia.

Ao final, a classe de `SEmanticsController` retorna para a camada de visão um relatório com as classes, atributos e métodos relevantes para a mudança e suas respectivas probabilidades de impactado.

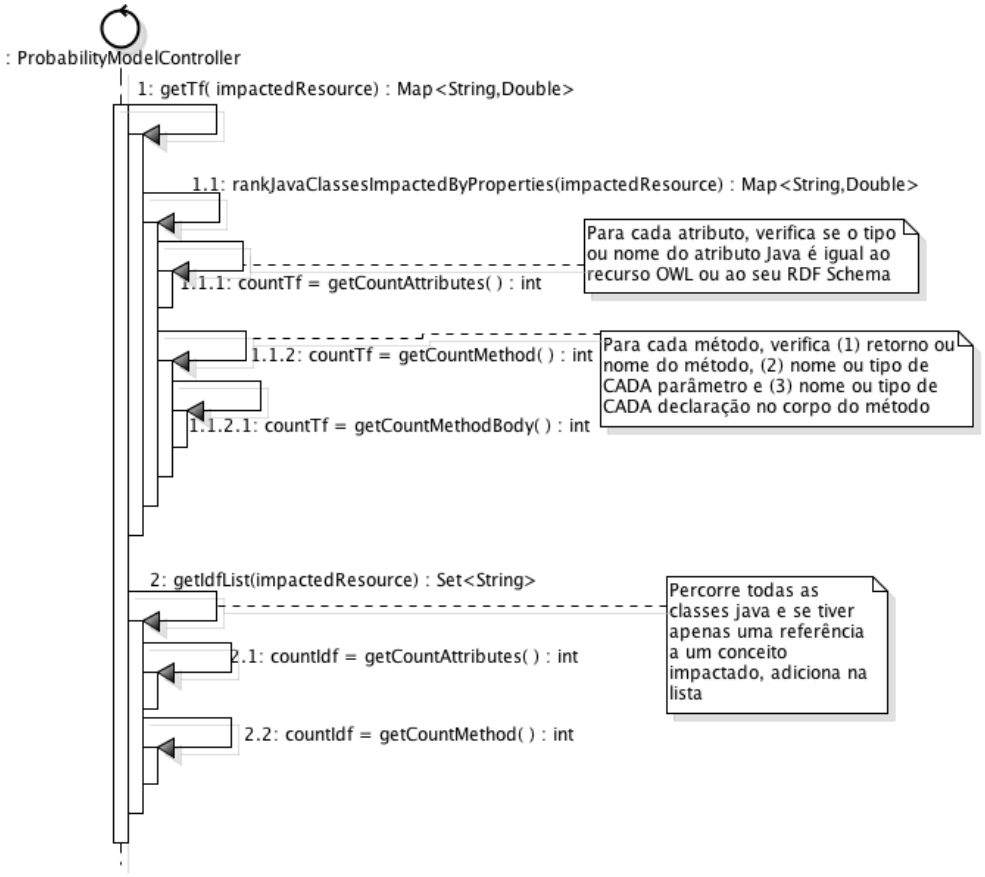


Figura 4.10 – Diagrama de sequência para o cálculo de TFIDF.

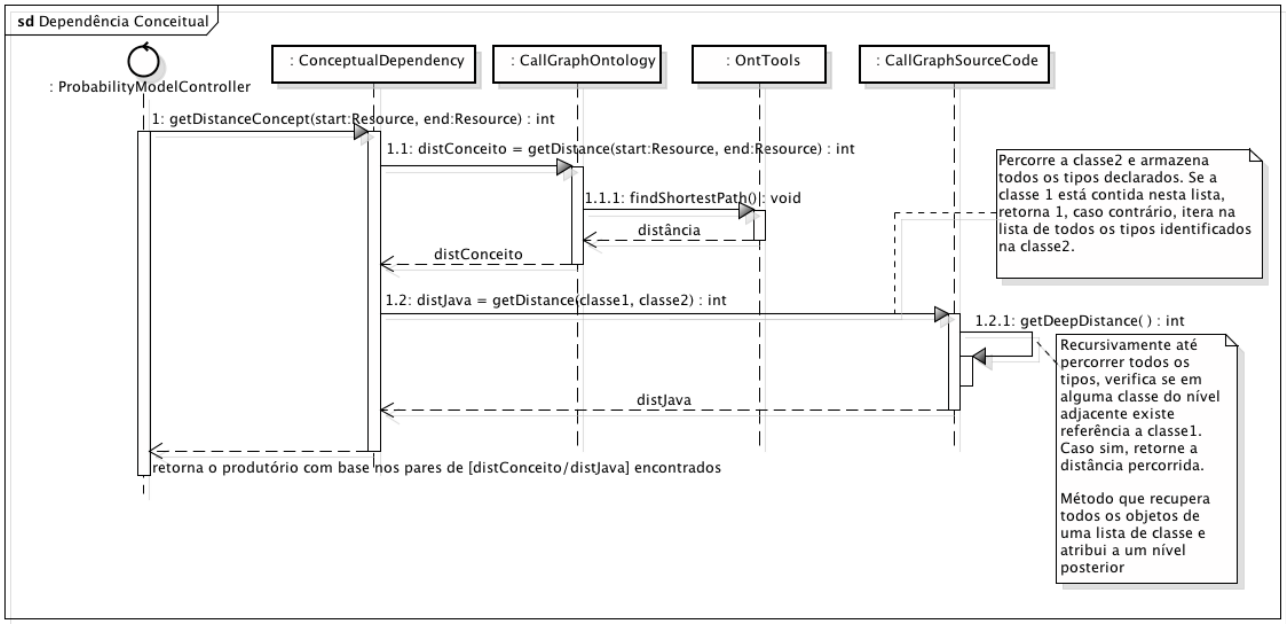


Figura 4.11 – Diagrama de sequência para o cálculo de DC.

4.2. Guia de Uso

Após a apresentação dos aspectos técnicos relacionados a solução da ferramenta SEmantics, é necessário fornecer um guia com a sequência de passos que devem ser executados para a utilização do modelo de análise de impacto desenvolvido.

Para a instalação do *plugin* no IDE Eclipse, basta copiar o arquivo *br.pucrs.semantics_1.0.0.jar* exportado do projeto SEmantics para a pasta de *plugins* da IDE e reiniciar. Uma vez carregado, os componentes do *plugin* são apresentados conforme Figura 4.12.

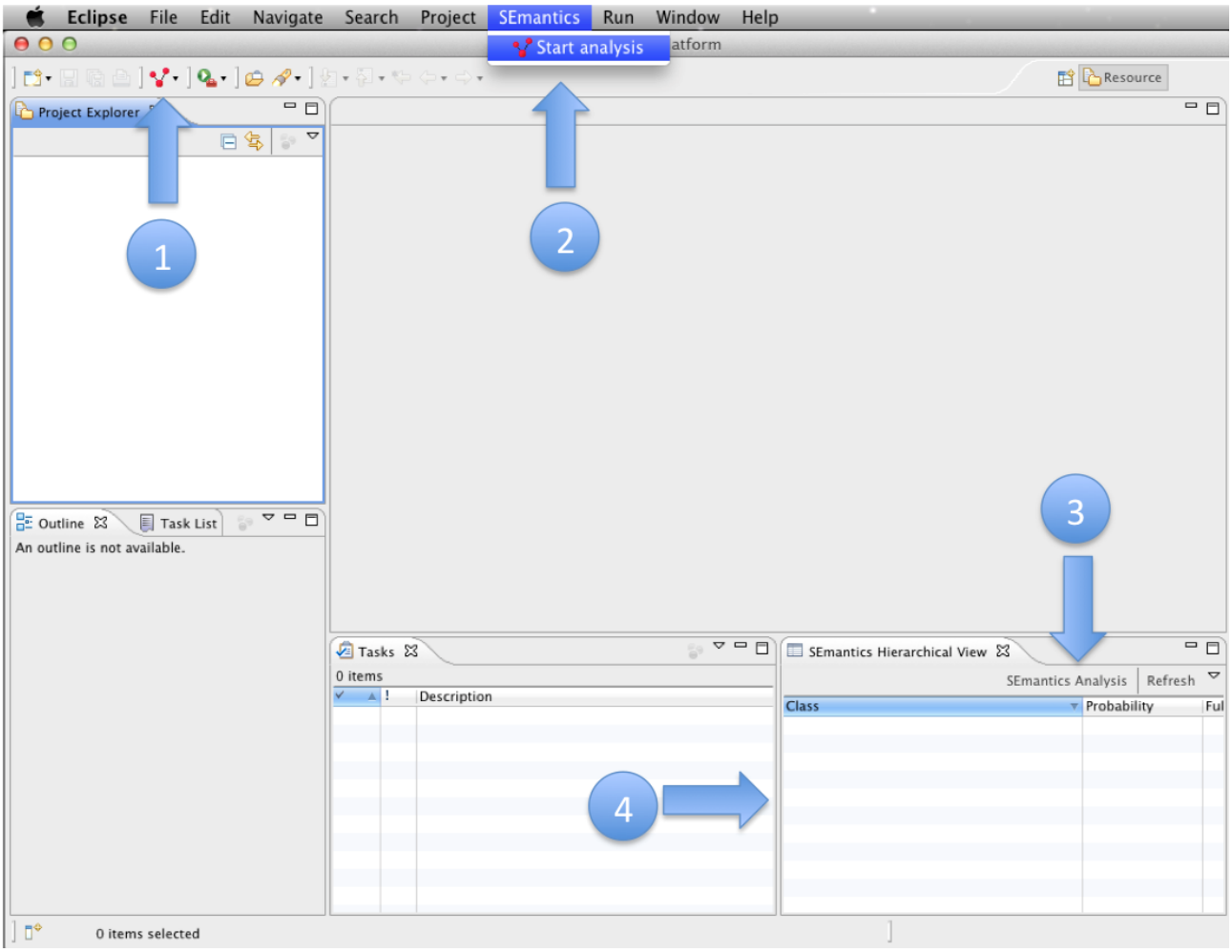


Figura 4.12 – Interface inicial do plugin SEmantics.

As marcações (1), (2) e (3) da Figura 4.12 representam formas alternativas de acessar a funcionalidade SEmantics. A marcação (4) apresenta o painel no qual o resultado da análise de impacto é fornecido ao usuário. Esta estrutura apresenta uma lista de classes hierarquizadas por métodos e atributos que são identificados como potencialmente impactados.

Para ilustrar um exemplo de uso, foi importado o projeto Timecard, explorado no Capítulo 3, utilizando o mecanismo padrão de importação de projeto do Eclipse (*File-Import*) e acessada a funcionalidade através do marcador (1) da Figura 4.12. O resultado é apresentado na Figura 4.13.

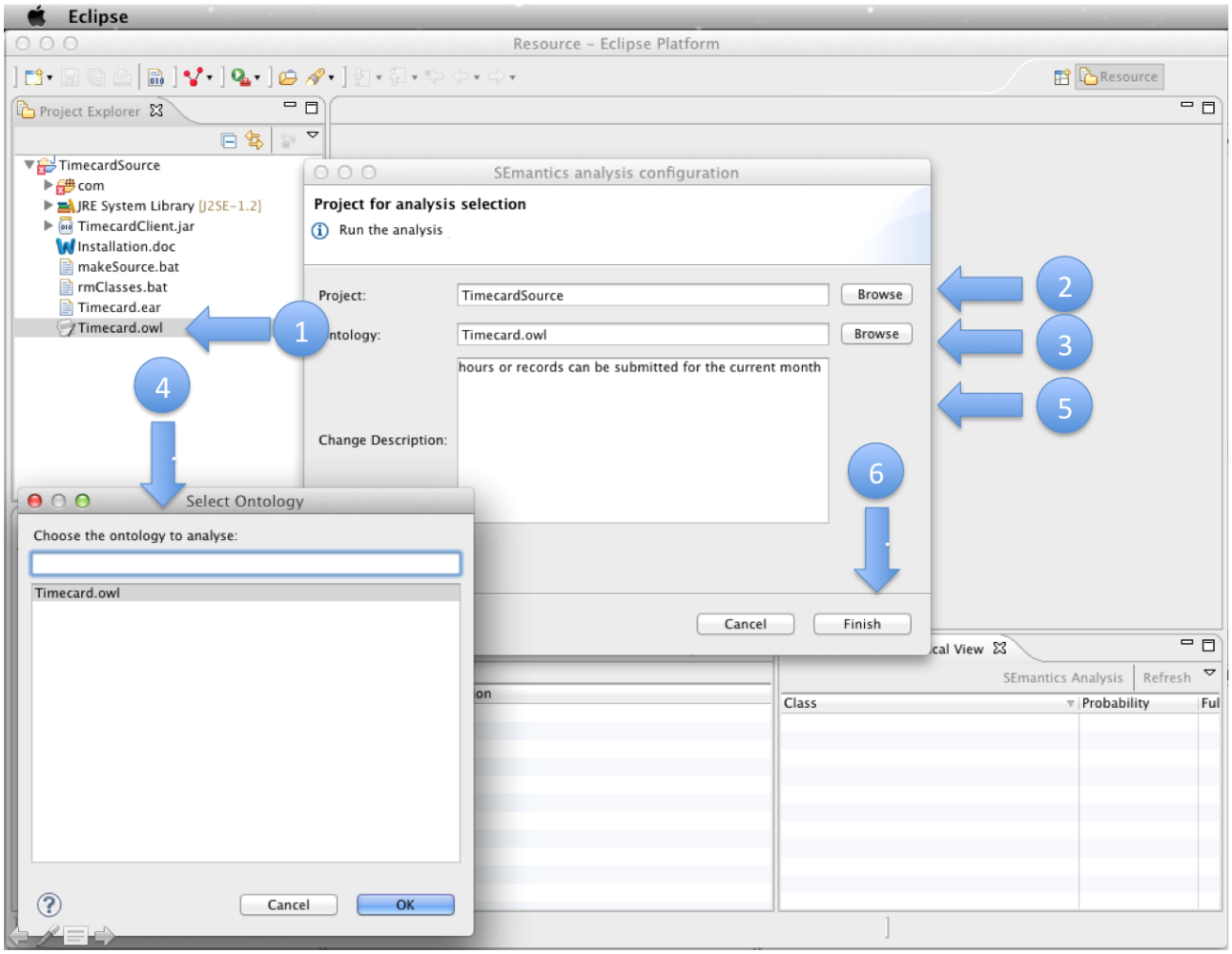


Figura 4.13 – Interface SEmantics para análise de impacto.

A Figura 4.13 apresenta a tela após a solicitação de análise de impacto. Note que o projeto importado possui uma ontologia chamada Timecard.owl conforme marcador (1). Ao solicitar a análise de impacto, a janela "*SEmantics analysis configuration*" é apresentada para seleção de projeto (2), ontologia (3) e descrição de mudança (5). Todos os projetos no *workspace* do Eclipse estão disponíveis para seleção ao clicar em *Browse* do marcador (2). Uma vez que o projeto foi escolhido, é possível selecionar qual ontologia deste projeto será utilizada para análise de impacto através da modal de seleção ilustrada pelo marcador (4). Esta modal percorre todo o projeto (2) e lista todos os arquivos OWL. Neste momento, o usuário pode realizar um filtro para buscar a ontologia pelo seu nome,

caso exista mais de uma. O mesmo mecanismo de filtro está disponível para seleção de projeto. Após seleção de projeto, ontologia e requisito de mudança, o usuário pode selecionar *Finish* (6) para executar a análise de impacto. O resultado é apresentado na Figura 4.14.

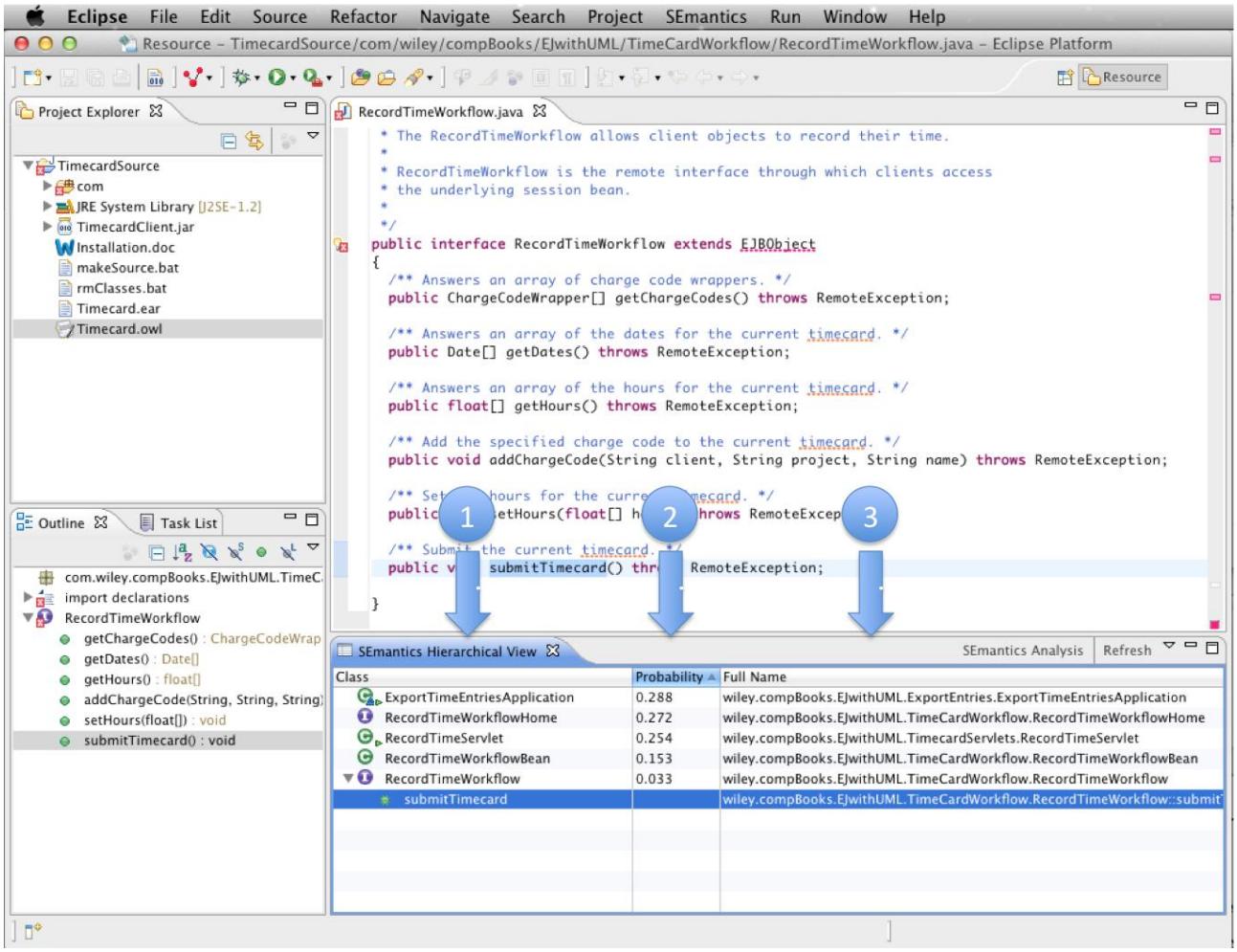


Figura 4.14 – Interface com os resultados da análise de impacto.

A Figura 4.14 apresenta os resultados da análise, informando as classes potencialmente impactadas (1), sua probabilidade de impacto (2) e seu caminho no projeto em questão (3). É possível expandir a árvore das classes caso seja identificado um método ou atributo potencialmente impactado, conforme ilustrado pela figura. Ao se clicar na classe, método ou atributo, os mesmos serão abertos para edição.

4.3. Considerações sobre o Capítulo

Esse capítulo descreveu os principais aspectos relacionados ao desenvolvimento da ferramenta SEmantics, que implementa o Modelo de Análise de Impacto proposto. Foram descritas duas visões: a estrutural e a comportamental.

A visão estrutural descreveu as APIs utilizadas, bem como detalhes de implementação. Foram discutidas as camadas da aplicação, os padrões de projeto adotados e a organização geral das classes. O propósito de cada componente da arquitetura também foi explorado em detalhes, bem como sua relevância para o Modelo de Análise de Impacto.

Toda a dinâmica associada à execução da análise de impacto foi apresentada pela visão comportamental, incluindo detalhes e fluxos dos algoritmos. Os processos dos Modelos de Rastreabilidade e Probabilístico foram descritos, bem como os eventos relacionados a análise de similaridade léxica e semântica.

Este capítulo também apresentou um guia de uso da ferramenta através da descrição do *plugin* desenvolvido para o IDE Eclipse. O cenário motivacional explorado pelo Capítulo 3 foi utilizado para ilustrar a dinâmica de uso da ferramenta. O modo de operação consiste em abrir no Eclipse um projeto Java e selecionar acesso a funcionalidade *SEmantics Analysis*. A partir deste momento, o usuário deve informar um projeto existente em seu *workspace*, uma ontologia pertencente a este projeto e a descrição da mudança. Informado estes dados, o *plugin* atualiza um painel na ferramenta com a relação de classes e suas respectivas probabilidades de impacto. Estas classes e seus membros ficam disponíveis para inspeção através de uma seleção direta, facilitando os procedimentos de manutenção.

5. AVALIAÇÃO EMPÍRICA

O modelo de análise de impacto em código fonte usando ontologias e recuperação de informação apresenta duas perspectivas para sua avaliação. A primeira está relacionada com a necessidade de desenvolver um produto de software, demonstrando sua viabilidade conforme apresentado no Capítulo 4. A segunda perspectiva está relacionada com a avaliação de sua eficiência e eficácia quando comparadaa abordagens manuais que representam o estado da prática, conforme discutido por [Jon07]. Conforme apresentado em Lindvall [Lin98], a análise manual do código fonte representa uma das estratégias mais comuns para analisar o impacto de mudanças. Para avaliar o modelo de análise de impacto proposto, se sugere um estudo comparando os resultados obtidos pela ferramenta SEmantics e por abordagens convencionais. Neste contexto, experimentos são determinantes para uma boa avaliação, provendo uma disciplinada, sistemática, quantificada e controlada forma de avaliar as atividades desempenhadas por humanos.

A literatura provê algumas abordagens baseadas em uma estratégia experimental para avaliar um processo onde o fator humano é considerado. Em [Pfl04], as seguintes abordagens são definidas para avaliação de processos, produtos e recursos: análise das características, pesquisa de opinião (*survey*), estudo de caso e experimentos. Esse último representa o tipo de estudo mais controlado, geralmente realizado em laboratórios. Nesta abordagem, os valores das variáveis independentes (entradas do processo) são manipulados para se observar as mudanças nos valores das variáveis dependentes (saídas do processo). Ao término da execução do experimento, os resultados são analisados, interpretados, apresentados e, por fim, empacotados.

Em [Kit04], é observado que as diferenças entre os métodos de pesquisa são refletidas em suas escalas. Experimentos tendem a ser pequenos, envolvendo um número reduzido de pessoas ou eventos devido a sua natureza que exige maior controle. Pode-se pensar em experimentos como “pesquisas em um ambiente restrito”. Por outro lado, estudos de caso geralmente abordam um projeto típico em vez de tentar obter informações sobre todos os possíveis casos; eles podem ser considerados como “pesquisas em um ambiente típico”.

O objetivo deste trabalho é investigar métodos alternativos para a análise de impacto, identificando relações como “mais preciso que”, “maior revocação que” ou “mais esforço que“. É possível, assim, isolar variáveis que determinam esta relatividade do resto

do processo e manipulá-las, avaliando os resultados a partir dos tipos de combinações possíveis. Para isso, torna-se necessário um controle sobre as variáveis independentes do experimento. Frente ao apresentado, optou-se pela utilização de experimentos para avaliar empiricamente o modelo proposto por este trabalho. Para conduzir o experimento, o processo aplicado utilizou como referência as propostas de [Woh00, Tra02], de maneira complementar. A avaliação do experimento foi apoiada por [Bis04, Ara06].

Este trabalho inclui três experimentos distintos. O primeiro experimento está relacionado à precisão e revocação dos elementos recuperados pelo modelo de análise de impacto proposto. O segundo está relacionado à análise de similaridade, que corresponde a um subprocesso chave da automação da análise de impacto. Essa análise é central em todo o modelo e corresponde ao processo que vincula código fonte aos conceitos da ontologia. A automação da análise de similaridade viabiliza o modelo de rastreabilidade utilizado para definição de seus elos, e o modelo probabilístico, utilizado para análise de probabilidade condicional que inclui TFIDF e DC. Devido à criticidade da análise de similaridade, a mesma foi considerada relevante para uma avaliação minuciosa. O terceiro experimento foi executado para caracterizar o esforço, avaliando o tempo necessário para analisar e implementar uma mudança utilizando ou não a ferramenta SEmantics.

5.1. Experimento sobre medida F associada à Análise de Impacto

5.1.1. Definição

A abordagem GQM (*Goal-Question-Metric*) [Bas94] foi utilizada para a definição deste experimento. O objetivo geral foi comparar o resultado da análise de impacto manual, realizada por engenheiros de software, com a obtida pela execução da ferramenta SEmantics. A partir deste objetivo, derivou-se a seguinte questão: *A abordagem automática para análise de impacto utilizando SEmantics (μaut) identifica o mesmo conjunto de classes do código fonte que a análise realizada manualmente por engenheiros de software (μman)?*

O objetivo da medição foi analisar o impacto identificando, em uma manutenção de software, quais as classes do código fonte relevantes para uma mudança. Para tanto, a métrica associada à questão previamente definida foi a medida F, que equivale a média harmônica entre a precisão, que mede quantos documentos relevantes foram recuperados, e a revocação, que mede a proporção de documentos relevantes

recuperados, conforme descrito na Seção 2.3.4. A definição das classes relevantes para a requisição de mudança foi definida por um especialista através da efetiva implementação das mudanças no código fonte da aplicação.

O objetivo do estudo foi definido como:

> *Comparar* a análise de impacto manual com a análise de impacto automatizada pela ferramenta SEmantics,
>
> *Com o propósito de* avaliar a eficiência de ambas as abordagens,
>
> *Com foco* na precisão e revocação,
>
> *Sob o ponto* de vista de um programador,
>
> *No contexto* de uma implementação de um requisito de mudança durante a manutenção evolutiva de software.

5.1.2. Planejamento e Execução

Foi escolhido o contexto de uma universidade para a condução do experimento. Embora diversos autores argumentem sobre a necessidade de conduzir experimentos em ambientes realistas [Sjo02], semelhantes aos encontrados na indústria, esta cooperação demanda custos e riscos não previstos no escopo desta pesquisa.

O processo para o experimento foi a abordagem na qual o conjunto de participantes executou o experimento em um ambiente controlado. Este experimento não foi executado durante o desenvolvimento de software industrial, isto é, ele foi *off-line*. Os participantes do experimento foram alunos do Programa de Pós Graduação em Ciência da Computação da PUCRS.

Com relação a realidade do sistema, foi utilizado um software de código aberto chamado Memoranda [Mem12], desenvolvido em Java para a gestão e organização pessoal de atividades, tarefas e recursos. O objetivo desta escolha foi a utilização de um sistema real, desenvolvido por terceiros sem a intervenção do pesquisador, e que seja diferente do exemplo utilizado durante a definição do trabalho, desacoplando assim a solução aqui proposta de um problema particular (como o sistema de cartão ponto).

Para o experimento, foi definida informalmente a hipótese de que a identificação de classes do código fonte resultantes da análise de impacto automatizada é igual a prática manual desenvolvida por engenheiros de software. Com base nessa definição

informal, viabiliza-se a formalização das hipóteses do experimento e suas medidas para avaliação.

A Hipótese Nula (**H_0**) definida foi que, utilizando um mesmo requisito de mudança (RdM), a medida F de ambas as abordagens (μaut e μman) é a mesma se comparadas a definição do impacto por um especialista que efetivamente implementou as modificações no código fonte, onde:

H_0: F μaut = F μman

H_1: F μman > F μaut

H_2: F μaut > F μman

Sendo que F (medida F) representa a média harmônica entre M_P (média da precisão) e M_R (média da revocação) do conjunto de classes relevantes e recuperadas pelos sujeitos (participantes) do experimento. A definição de F se dá conforme:

$$F = \frac{2}{1/M_P + 1/M_R}$$

Onde M_P e M_R são definidos conforme:

$$M_P = \frac{\sum_{s=1}^{S} Precisão(s)}{S}$$

$$M_R = \frac{\sum_{s=1}^{S} Revocação(s)}{S}$$

Sendo S a quantidade de sujeitos s participantes do experimento. As fórmulas de precisão e revocação estão descritas na Seção 2.3.4.

Para o experimento, assumiu-se como variável independente a técnica para análise de impacto e como variável dependente a corretude na definição do impacto da mudança no código fonte utilizando a medida F. A seleção dos sujeitos participantes foi por conveniência (não probabilística) e incluiu dezoito (18) profissionais graduados e com experiência prévia em desenvolvimento de software.

Dentre os princípios genéricos de projeto do experimento, foi avaliada a obstrução, na qual nem todos os participantes possuem conhecimento equivalente em Java. Para minimizar o efeito da experiência, a análise do impacto foi realizada em duplas, nas quais os participantes foram agrupados por conveniência e quota pelo condutor do experimento, utilizando como critério suas experiências na linguagem Java.

O tipo de projeto do experimento foi um fator (medida F) com dois tratamentos (µaut e µman). Para a realização dos testes das hipóteses no contexto do experimento (um fator e dois tratamentos aleatórios), a literatura sugere o teste de significância chamado Teste T para duas amostras independentes, caso seja realizado um teste paramétrico, ou Mann-Whitney, caso o teste seja não paramétrico. A definição do teste a ser aplicado ocorrerá após a análise da normalidade e variância dos dados obtidos pela execução do experimento.

A instrumentação do experimento incluiu como objeto o código fonte da aplicação Memoranda, a IDE Eclipse para análise do código e três requisições de mudanças. Como guia para µman, a equipe foi motivada e treinada, sendo apresentada a arquitetura do software e como importar e executar a aplicação. Também foi fornecido um documento sobre como proceder durante o experimento. As métricas foram coletadas em formulários preenchidos pelos participantes (Relatório de Análise de Impacto) conforme Figura 5.1. Para a abordagem µaut, a análise de impacto foi executada utilizando a ferramenta SEmantics.

Relatório de Análise de Impacto

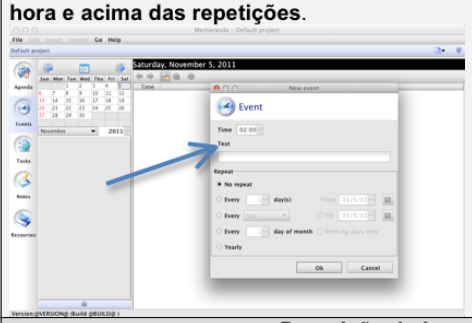
Nomes	
Hora de Início	
Hora de Término	
Análise de Impacto no Projeto	
Descrição da Alteração	**Permitir ao usuário informar a data de um evento logo abaixo do texto e hora e acima das repetições.** 
Código Fonte Impactado	**Descrição do Impacto**
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
Observações	

Figura 5.1 – Relatório de Análise de Impacto.

As requisições de mudanças foram definidas em língua Portuguesa para µman e em língua Inglesa para µaut, conforme Tabela 5.1. Foi utilizada a língua Portuguesa para µman com o objetivo de facilitar o entendimento da mudança, eliminando o viés da língua estrangeira. A língua inglesa foi utilizada em decorrência do *parser* realizado pelo modelo.

A instrumentação completa está disponível no Apêndice E, junto com a ontologia utilizada para abordagem µaut.

Tabela 5.1 – Requisições de mudanças para ferramenta memoranda.

µman	µaut
Permitir ao usuário informar a data de um evento logo abaixo do texto e hora e acima das repetições.	User can add event description below text, time and above norepeat.
O usuário pode adicionar uma descrição do evento abaixo do texto e hora e acima das repetições.	User can set up event date below text, time and above norepeat.
Adicionar recursos a janela de tarefas próximo ao TODO, descrição, esforço, datas, notificação, prioridade e progresso.	Add resource to taskdialog next to todo, description, effort, dates, notification, priority and progress.

Com relação a análise da validade interna, alguns critérios foram considerados, tais como: (1) histórico, na qual a data de aplicação do experimento foi criteriosamente definida para evitar períodos nos quais os participantes pudessem sofrer influências externas, como final de semestre; (2) maturação, para motivar positivamente os participantes durante o treinamento; (3) seleção dos grupos, utilizado para nivelar o conhecimento dos participantes através de treinamento e agrupamento conforme experiência; e (4) difusão, para incentivar os participantes a não interagirem entre os grupos, aliado ao policiamento do condutor do experimento.

A validade externa incluiu a interação da seleção, através da identificação de participantes com perfil apto ao tratamento do experimento, apresentando, em sua maioria, conhecimento prévio de programação e experiência na indústria.

A validade de construção considerou a explicação operacional do experimento, visando maturar a forma na qual a análise de impacto foi conduzida e sua extração de dados. A expectativa do condutor do experimento também foi considerada, fazendo com que o mesmo não exerça influência sobre as variáveis envolvidas ou sobre o material elaborado.

A validade da conclusão considerou as seguintes perspectivas:

- Manipulação dos dados: como os dados resultantes do experimento serão manipulados pelo pesquisador, é possível que os mesmos sofram algumas variações, tal como o coeficiente de significância para validação dos resultados;

- Confiabilidade das medidas: esta perspectiva sugere que medidas subjetivas possam ser influenciadas pelo pesquisador. Em nossa proposta, as medidas foram objetivamente definidas, não dependendo do critério humano;
- Confiabilidade na implementação dos tratamentos: consiste no risco em que diferentes participantes possam implementar de forma distinta os processos estabelecidos pelo experimento. Este risco não será evitado, visto que não se pode interferir no caráter subjetivo de analisar os impactos de determinada mudança;
- Configurações do ambiente do experimento: consiste nas interferências externas do ambiente que podem influenciar os resultados durante a execução do experimento. O experimento foi executado em um laboratório isolado, onde foi proibida a interação externa como celulares, saídas, etc.;
- Heterogeneidade aleatória dos participantes: a escolha de diferentes participantes com diferentes experiências pode exercer um risco na variação dos resultados.

Para preparar a execução do experimento, os participantes consentiram com o experimento, pois se os participantes não concordam com os objetivos da pesquisa ou não tem conhecimento sobre o experimento, corre-se o risco de que sua participação não ocorra em encontro aos objetivos. Durante a experimentação, a preparação dos participantes forneceu o embasamento necessário sobre o experimento, esclarecendo os objetivos e metas almejadas.

Resultados sensitivos também foram considerados, pois é possível que o resultado obtido pelo experimento seja influenciado por questões pessoais, como a sensibilidade dos participantes por estarem sendo avaliados. Foi adotada uma postura de anonimato em toda a execução do experimento.

Com relação à instrumentação, todas as definições apresentadas foram criteriosamente estabelecidas durante a execução, incluindo o ambiente e infraestrutura. Foi realizado um treinamento específico de como proceder a análise de impacto, contextualizando os objetivos, a técnica, a motivação e o procedimento operacional.

A execução do experimento foi realizada em um curto período de tempo (3 horas), no qual o pesquisador esteve envolvido em todos os detalhes da execução. Durante este período, foi realizada a coleta dos dados pelos próprios participantes, utilizando formulário

específico. Esta opção é justificada, pois o pesquisador não poderá se envolver na definição dos dados resultantes do experimento. Outro critério considerado foi o anonimato, onde os nomes dos participantes não foram divulgados. Durante a execução do experimento, o responsável ficou a disposição dos participantes para o esclarecimento das dúvidas relacionadas ao processo, abstendo-se de influência técnica nas respostas.

5.1.3. Análise

A primeira análise apresentada diz respeito à classificação das escalas das variáveis definidas no experimento, apresentada na Tabela 5.2. Com esta classificação, é possível determinar as operações que podem ser aplicadas sobre as variáveis.

Tabela 5.2 – Escalas das variáveis.

Variável	Nome	Escala
Dependente	Medida F	Razão
Independente	Técnica de Análise de Impacto	Nominal

O experimento obteve como resultado os dados brutos apresentados na Tabela 5.3 e sumarizados na Tabela 5.4.

Tabela 5.3 – Tabulação dos valores brutos do experimento.

RdM	Sujeito	Precisão	Revocação	Medida F
RdM 01	P1	0,50	1,00	0,67
	P2	0,50	1,00	0,67
	P3	0,50	1,00	0,67
	P4	0,33	1,00	0,50
	P5	0,20	1,00	0,33
	P6	0,50	1,00	0,67
	P7	0,50	1,00	0,67
	P8	0,33	1,00	0,50
	P9	1,00	1,00	1,00
	M(µman)	**0,49**	**1,00**	**0,63**
	SEmantics	0,50	1,00	0,67
	M(µaut)	**0,50**	**1,00**	**0,67**
RdM 02	P1	1,00	0,80	0,89
	P2	1,00	0,60	0,75
	P3	-	-	-
	P4	-	-	-

	P5	-	-	-
	P6	1,00	0,40	0,57
	P7	1,00	0,40	0,57
	P8	1,00	0,60	0,75
	P9	1,00	0,20	0,33
	M(µman)	**1,00**	**0,50**	**0,64**
	SEmantics	1,00	0,80	0,89
	M(µaut)	**1,00**	**0,80**	**0,89**
RdM 03	P1	0,00	0,00	0,00
	P2	0,50	1,00	0,67
	P3	-	-	-
	P4	-	-	-
	P5	-	-	-
	P6	-	-	-
	P7	0,33	1,00	0,50
	P8	1,00	1,00	1,00
	P9	1,00	1,00	1,00
	M(µman)	**0,57**	**0,80**	**0,63**
	SEmantics	1,00	1,00	1,00
	M(µaut)	**1,00**	**1,00**	**1,00**

Tabela 5.4 – Tabulação dos valores sumarizados do experimento.

Sujeito	Abordagem	Precisão	Revocação	Medida F
RdM 01	M(µman)	0,49	1,00	0,63
	M(µaut)	0,50	1,00	0,67
RdM 02	M(µman)	1,00	0,50	0,64
	M(µaut)	1,00	0,80	0,89
RdM 03	M(µman)	0,57	0,80	0,63
	M(µaut)	1,00	1,00	1,00

A Figura 5.2 representa o gráfico de linhas da medida F associada a cada abordagem de análise de impacto. Todas as análises apresentadas neste experimento foram realizadas utilizando o pacote estatístico SPSS [Sps12].

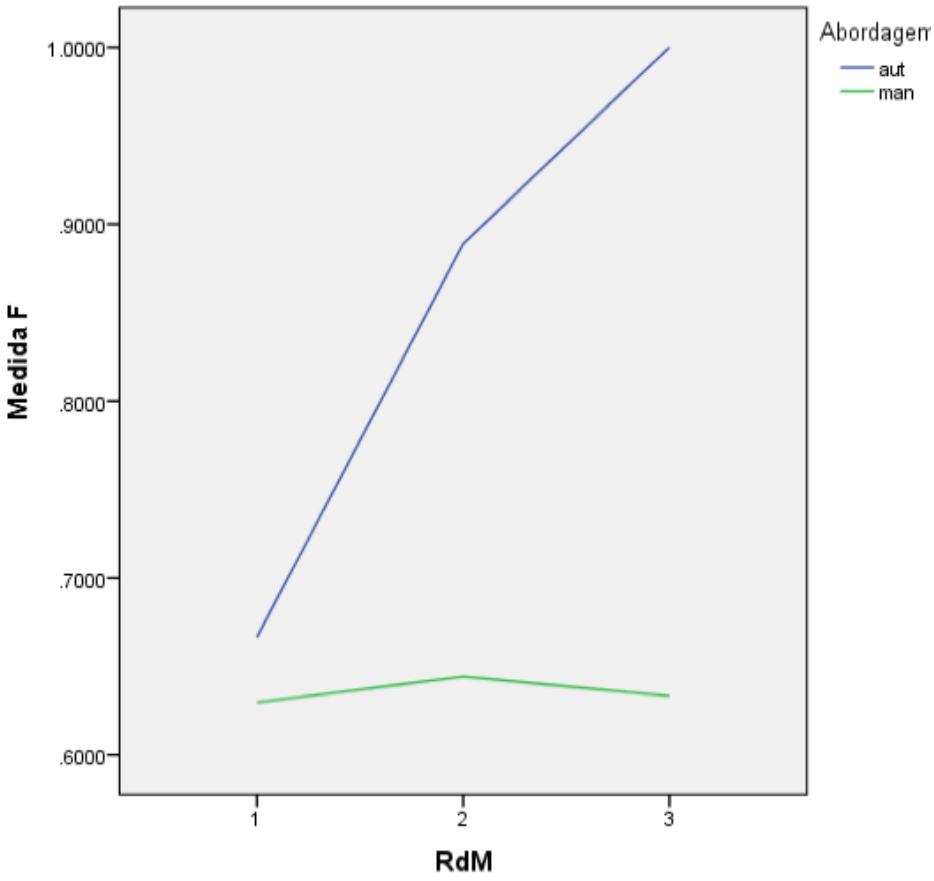


Figura 5.2 – Gráfico de Linhas com o resultado da execução.

Conforme apresentado, as variáveis dependentes estão caracterizadas na escala razão, o que permite o cálculo da normalidade e homocedasticidade necessárias para definir o tipo de teste das hipóteses (paramétrico ou não paramétrico).

Para a análise da hipótese, os dados foram caracterizados, visualizando tendências centrais e dispersões. Posteriormente, sugere-se a eliminação de dados anormais ou incertos, que distorcem a integridade da conclusão, através da redução do intervalo de dados. Por último, foi realizado o teste das hipóteses que compreende a avaliação estatística dos dados até certo nível de significância. O nível de significância adotado (p-value) para todos os testes foi de 5%. O p-value compreende o menor nível de significância com que se pode rejeitar a hipótese nula.

Uma análise inicial da distribuição é eficiente para avaliar o comportamento das amostras. Foi utilizado o gráfico de dispersão *boxplot*, apresentado na Figura 5.3, para identificação dos *outliers*.

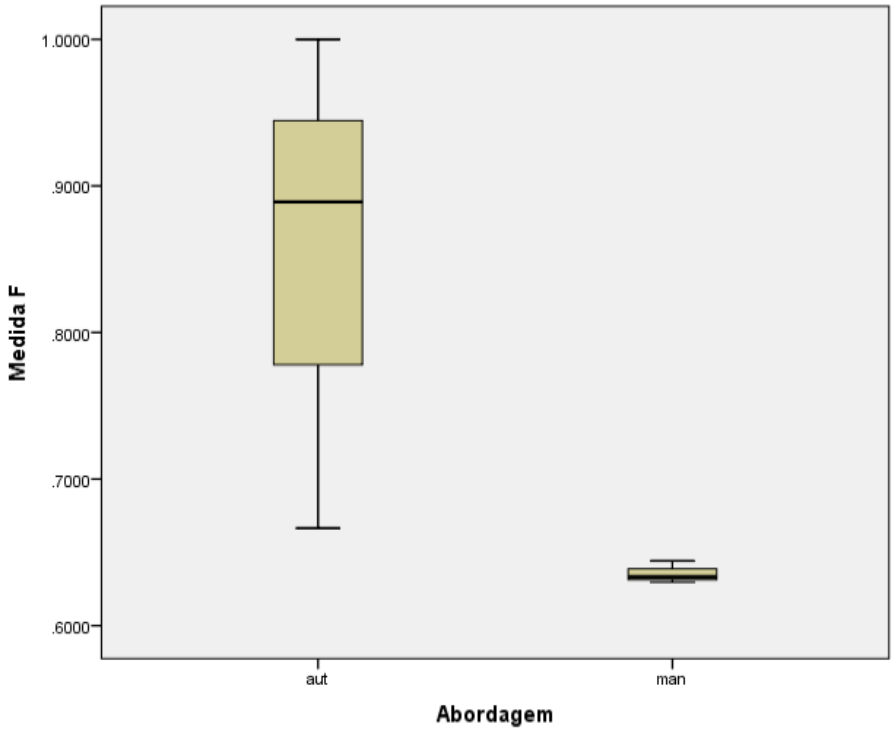


Figura 5.3 – Gráfico de dispersão.

Conforme apresentado na Figura 5.3, a variável medida F possui *outliers* moderados e portanto nenhum sujeito foi eliminado da amostra para o teste de hipótese. Outra observação sobre o gráfico de dispersão é a variabilidade entre as duas amostras, na qual a dispersão da abordagem manual é muito pequena.

A próxima etapa consiste em identificar se os dados seguem uma distribuição normal. Para se avaliar a normalidade, é definida uma hipótese nula e uma hipótese alternativa, conforme:

H_0: A distribuição é normal;

H_1: A distribuição não é normal.

Existem duas formas para se avaliar a distribuição normal dos dados, que compreendem o Teste de Kolmogorov-Smirnov e o Teste de Shapiro-Wilk. O primeiro é utilizado para identificar a normalidade em variáveis com pelo menos 30 valores e o segundo em variáveis com menos de 50 valores. A Tabela 5.5 apresenta os testes de normalidades para a amostra utilizando o Teste de Shapiro-Wilk.

Tabela 5.5 – Teste de normalidade Shapiro-Wilk para variável Medida F.

Abordagem	Estatística	Grau de Liberdade	Significância
μaut	0,964	3	0,637
μman	0,926	3	0,473

Com base na Tabela 5.5, observa-se que a significância dos dados do teste de Shapiro-Wilk é superior, em ambas as amostras (μman e μaut), ao nível de significância definido (0,05 ou 5%). Com esta informação, não há indícios para rejeitar a hipótese nula sobre a distribuição da normalidade, conseguindo assim o primeiro requisito para utilização de teste paramétrico para duas amostras independentes.

O segundo requisito requer a análise da homocedasticidade, tornando necessário analisar a variância das duas amostras. Com este objetivo, definem-se duas hipóteses:

H_0: As variâncias são iguais;

H_1: As variâncias não são iguais.

O teste das hipóteses acima é realizado com a significância obtida diretamente do Teste de Levene. O Teste de Levene é usado para testar se *k* amostras têm a mesma variância. A Tabela 5.6 apresenta os resultados obtidos para este teste.

Tabela 5.6 – Teste de Levene para igualdade das variâncias sobre medida F.

	Significância
Assumindo variâncias iguais	0,057
Não assumindo variâncias iguais	

Com base na Tabela 5.6, verifica-se que o nível de significância para variâncias iguais (0,057) é superior ao nível de significância definido. Com esta informação, não se consegue rejeitar a hipótese nula para variâncias, conseguindo o segundo requisito para utilização do teste paramétrico.

Conforme definido no planejamento do projeto do experimento, o teste indicado para avaliação das hipóteses é o Teste T para duas amostras independentes. Consegue-se validar sua utilização por ser um teste paramétrico, no qual seus requisitos foram explicitamente atendidos.

Com base na declaração das hipóteses, têm-se:

H_0: F μaut = F μman

H_1: F μman > F μaut

H_2: F μaut > F μman

O critério para rejeição de H_0 em favor de **H_1** é:

H_1: (F μman > F μaut): rejeita-se H_0 se $t_0 > t_{\alpha, n+m-2}$, onde

t_0: é o valor t obtido através da aplicação do Teste T.

$t_{\alpha, n+m-2}$: é o valor obtido pela tabela de Distribuição de T, apresentada no Anexo A, onde *n+m–2* representa o grau de liberdade, sendo *n* o número de participantes de uma abordagem e *m* o número de participantes de outra abordagem. O grau de liberdade de nossa amostra é 8.

O Teste T para duas amostras independentes foi aplicado neste contexto e o resultado está apresentado na Tabela 5.7.

Tabela 5.7 – Teste T para medida F agrupado por μman e μaut

	T	**Grau de Liberdade**	**Significância (bicaudal)**
Assumindo variâncias iguais	-2,404	4	0,042
Não assumindo variâncias iguais	-2,404	2,008	0,158

Com base na Tabela 5.7, obtemos o valor de t_0 (= -2,404) e, com base no Anexo A, obtemos o valor de $t_{\alpha, n+m-2}$ (= 2,31). Como $t_0 < t_{\alpha, n+m-2}$, então não se consegue rejeitar a hipótese nula a um nível de significância de 5% em favor de **H_1**: F μman > F μaut.

O critério para rejeição de **H_0** em favor de **H_2** é:

H_2: (F μaut > F μman): rejeita-se **H_0** se $t_0 > t_{\alpha, n+m-2}$

O Teste T foi aplicado neste contexto e o resultado está descrito na Tabela 5.8.

Tabela 5.8 – Teste T para medida F agrupado por μaut e μman

	T	**Grau de Liberdade**	**Significância (bicaudal)**
Assumindo variâncias iguais	2,404	4	0,042
Não assumindo variâncias iguais	2,404	2,008	0,158

Com base na Tabela 5.8, obtemos o valor de t_0 (= 2,404) e o valor de $t_{\alpha, n+m-2}$ (= 2,31). Como $t_0 > t_{\alpha, n+m-2}$, consegue-se rejeitar a hipótese nula a um nível de significância de 5% em favor de **H_2**: F μaut > F μman.

Pelas análises apresentadas, pode-se concluir que existe diferença estatisticamente significativa com relação às estruturas de código fonte recuperadas automaticamente pela ferramenta SEmantics comparada a análise manual do impacto. O desempenho obtido pela medida F associada a análise de impacto automática é maior

que a análise de impacto manual. Isso representa que a ferramenta, no contexto deste experimento, recupera mais documentos relevantes (precisão) e possui uma melhor proporção desses documentos recuperados (revocação) se comparada a análise manual de engenheiros de software.

5.1.4. Interpretação

O ponto fundamental para a automação da análise de impacto é a habilidade de identificar quais as estruturas de código fonte impactadas por uma mudança. Este experimento foi realizado com o intuito de avaliar o quão próximo da identificação de código relevante é o resultado obtido pela automação do modelo de análise de impacto em código fonte usando ontologia proposto comparado a uma estratégia manual de análise impacto. Para esta avaliação, foi utilizada a medida F definida pela média harmônica entre precisão, que mede quantas classes relevantes foram recuperadas, e revocação, que mede a proporção de classes relevantes recuperadas.

Os dados relativos à medida F obtiveram uma distribuição normal com uma diferença estatisticamente insignificante entre suas variâncias (homocedasticidade). Diante desta circunstância, foi executado o Teste Paramétrico T que evidenciou a validade da hipótese alternativa que estabelece que a medida F da análise de impacto automática é maior que a obtida manualmente por engenheiros de software. Como resultado, pode-se comprovar estatisticamente a hipótese de que a precisão e a revocação da análise de impacto automática definida pela ferramenta SEmantics é maior que a recuperada pela análise de impacto manual.

Um ponto fundamental para análise de impacto é a análise de similaridade que automatiza a identificação dos elos de rastreabilidade, população automática da ontologia e análise da probabilidade condicional necessária para ponderar cada elo. Esta análise de similaridade corresponde a habilidade de identificar quais conceitos do domínio estão presentes em uma especificação de software.

Este mapeamento não é trivial e idealmente deveria ser executado por um especialista do domínio, discernindo e relacionando a perspectiva conceitual das demais perspectivas de um sistema de informação. O esforço associado a esta identificação é significativo e não é esperado que, durante o processo de desenvolvimento de software, pessoas relacionem manualmente estes conceitos a cada elemento que constituem os

artefatos, principalmente um especialista do domínio que desempenha um papel chave neste processo.

Para avaliar aspectos relacionados a precisão e revocação da ferramenta desenvolvida para análise de similaridade comparada a uma avaliação humana e não especialista, que tipicamente caracterizaria desenvolvedores de software, foi realizado um experimento controlado conforme descrito a seguir.

5.2. Experimento sobre Análise de Similaridade associada à Análise de Impacto

5.2.1. Definição

A abordagem GQM também foi utilizada para a definição do estudo. Tendo como base a possibilidade de melhorar o processo de identificação de conceitos em artefatos de software, o objetivo deste estudo foi avaliar a precisão e revocação da ferramenta que analisa a similaridade entre termos, comparando-a com a atividade humana não especialista de capturar e associar estas mesmas estruturas.

A partir deste objetivo, derivou-se a seguinte questão: *a precisão e revocação da definição de elos de rastreabilidade entre artefatos e conceitos utilizando μaut (análise de similaridade automática) é a mesma que utilizando μman (análise de similaridade manual) se comparadas à definição realizada por um especialista do domínio*? A métrica utilizada para responder esta questão foi a medida F definida no experimento anterior.

O objetivo de estudo foi definido como:

> *Comparar* a identificação manual de conceitos obtido por engenheiros de software não especialistas no domínio da aplicação, com o mapeamento automático obtido pelo processo proposto de análise de similaridade,
>
> *Com o propósito de* avaliar a eficiência de ambas as abordagens,
>
> *Com foco* na precisão e revocação,
>
> *Sob o ponto de vista de* um especialista do domínio,
>
> *No contexto* da definição de elos de rastreabilidade entre conceitos do domínio e artefatos de software.

5.2.2. Planejamento e Execução

O experimento foi conduzido em uma universidade, utilizando um ambiente controlado e *off-line*. A população definida para o experimento foi formada por estudantes dos cursos de graduação e pós-graduação em computação. A amostra da população foi não probabilística, escolhida por quota e conveniência.

A questão previamente definida originou a seguinte Hipótese Nula (**H_0**): utilizando um mesmo artefato, a medida F de ambas as abordagens é a mesma se comparada a identificação dos conceitos por um especialista, onde:

H_0: F μaut = F μman

H_1: F μman > F μaut

H_2: F μaut > F μman

As mesmas métricas definidas para o primeiro experimento foram utilizadas para avaliar a análise de impacto associada aos modelos de rastreabilidade e probabilístico. As considerações sobre as análises de validade interna, externa, de construção e conclusão também foram aplicadas nesta execução do experimento.

A instrumentação do experimento utilizou seis casos de uso (CdU) descritos na língua inglesa relacionado ao sistema de Cartão Ponto discutido no Capítulo 3. Como o objetivo da rastreabilidade é identificar conceitos em artefatos, a motivação em utilizar casos de uso e não o código fonte foi pela facilidade na interpretação e identificação dos conceitos pelos participantes nessa especificação. Como casos de uso são escritos em linguagem natural, o viés da questão técnica associada ao conhecimento da linguagem de programação seria minimizado, focando essencialmente na eficiência da definição de elos. A relação entre as dimensões de conceitos do domínio e casos de uso foi definida utilizando um formulário de mapeamento, conforme Figura 5.4.

USE CASE	**Administrative User**	**Charge Code**	**Client**	**Employee**
UC01-Create Charge Code				

Figura 5.4 – Formulário parcial para o mapeamento de descrição de casos de uso com conceitos do domínio.

As variáveis independentes foram μaut e μman e a variável dependente foi a medida F obtida em cada descrição de caso de uso. O projeto experimental utilizado foi de um fator com dois tratamentos.

Para a abordagem μaut, a análise de similaridade do Modelo de Rastreabilidade foi executada utilizando os artefatos descritos. Para a abordagem μman, foi oferecido um treinamento, apoiando e motivando os participantes para a execução do experimento. Para coleta dos dados, os participantes preencheram o formulário representado pela Figura 5.4. A instrumentação completa do experimento, incluindo treinamento e casos de uso, é apresentada no Apêndice F.

Antes da execução do experimento, as variáveis independentes e dependentes foram cuidadosamente preparadas, atentando para o consenso com o experimento, resultados sensitivos e anonimato. A execução do treinamento teve como objetivo contextualizar o estudo, motivar os indivíduos e demonstrar o modo operacional para execução do experimento.

O experimento possuiu 13 participantes que receberam a descrição de seis casos de uso e a relação completa dos conceitos do domínio referentes aos mesmos casos de uso. Foi solicitado que cada participante interpretasse os requisitos e avaliasse quais seriam os conceitos presentes nestas especificações. Após a execução do experimento, obtiveram-se os resultados apresentados a seguir.

5.2.3. Análise

Para este experimento, as variáveis independentes foram caracterizadas na escala nominal e as dependentes na escala razão. A escala razão permite avaliar a distribuição dos dados e a definir qual teste de hipótese pode ser aplicado (paramétrico ou não paramétrico). O nível de significância adotado (p-value) para todos os testes foi de 5%. A execução do experimento produziu os dados apresentados no Apêndice G e sumarizados pela Tabela 5.9.

Tabela 5.9 – Conjunto de Dados da Execução do Experimento.

Sujeito (CdU)	Abordagem	Precisão	Revocação	Medida F
1	μman	1,00	0,79	0,88
	μaut	1,00	1,00	1,00
2	μman	0,96	0,79	0,85
	μaut	1,00	1,00	1,00
3	μman	1,00	0,72	0,83
	μaut	1,00	1,00	1,00
4	μman	1,00	0,45	0,61
	μaut	1,00	0,75	0,86

5	μman	0,76	0,88	0,81
	μaut	0,67	1,00	0,80
6	μman	0,76	0,88	0,79
	μaut	1,00	1,00	1,00

A análise inicial teve como objetivo avaliar a distribuição dos dados. Para tanto foi gerado o gráfico de linhas conforme Figura 5.5 e o gráfico de *boxplot* conforme Figura 5.6. Mesmo identificando o *outlier* 07 (equivalente ao CdU 4 de μman) na Figura 5.6, nenhum sujeito foi retirado da amostra para o teste de hipóteses. Em uma amostra de apenas seis pontos, a eliminação de um sujeito não deve ser feita apenas com base em gráficos de *boxplot*, além de que nenhuma evidência externa foi identificada durante a execução do experimento que justificasse esta exclusão.

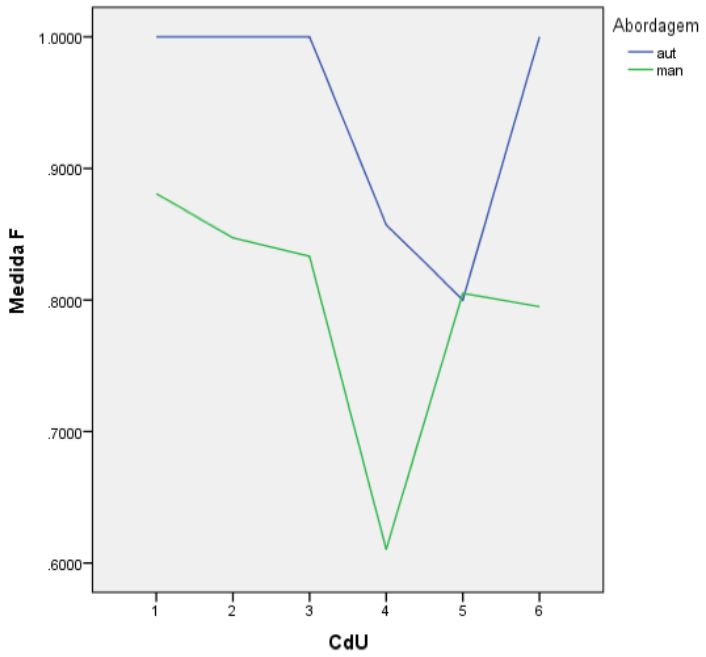


Figura 5.5 – Gráfico de Linhas com o resultado da execução.

O próximo passo foi identificar se os pontos seguem uma distribuição normal, onde as duas hipóteses relacionadas a normalidade definidas no experimento anterior foram consideradas. A Tabela 5.10 apresenta os testes de normalidades para a amostra utilizando o Teste de Shapiro-Wilk.

Tabela 5.10 – Teste de normalidade Shapiro-Wilk.

Variável	**Abordagem**	**Estatística**	**Grau de Liberdade**	**Significância**
Medida F	μaut	0,697	6	0,006
	μman	0,798	6	0,056

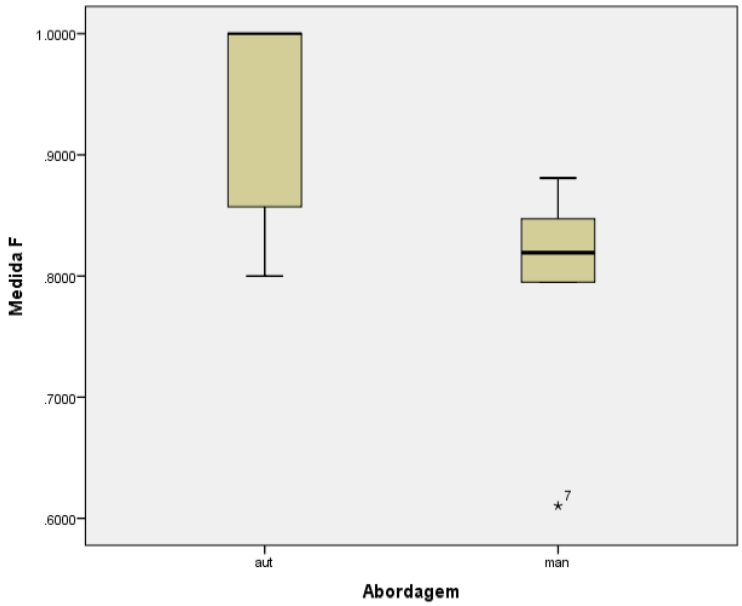


Figura 5.6 – Gráfico de dispersão.

Com base na Tabela 5.10, observa-se que a significância dos dados do teste de Shapiro-Wilk é inferior em μaut (0,006) com relação ao nível de significância definido (0,05 ou 5%). Sendo assim, há indícios para rejeitar a hipótese nula e, consequentemente, não se pode aplicar um teste paramétrico para avaliação das hipóteses. Optou-se por aplicar o teste Mann-Whitney, para duas amostras independentes, por se tratar de uma alternativa não paramétrica para o Teste T. Com base na declaração das hipóteses, sugere-se:

H_0: Não há diferença entre as médias (μaut = μman)

H_1: Há diferença entre as médias (μaut ≠ μman)

O resultado do teste Mann-Whitney foi aplicado sobre as amostras e está apresentado na Tabela 5.11.

Tabela 5.11 – Teste Mann-Whitney.

Variável	U de Mann-Whitney	W de Wilcoxon	Z	Sig. Assimpt. (bilateral)	Sig. Exata [2*(Sig.Unilateral)]
Medida F	5,000	26,000	-2,119	0,034	0,041a

(a) Não corrigidos para os empates

Como o grau de significância associado (Sig. Assimpt.) é 0,034 e é menor que a significância assumida de 0,05, pode-se rejeitar **H_0**. Frente aos resultados apresentados para a variável medida F, existe diferença de média entre a abordagem manual e automática. Pela análise estatística dos dados, consegue-se recuperar duas informações:

- A distribuição da medida F não é normal, o que implica na execução de testes não paramétricos;
- Utilizando o teste Mann-Whitney, conseguiu-se verificar que existem diferenças entre as médias das duas amostras μaut e μman.

Utilizando o teste de Mann-Whitney, conseguiu-se apenas rejeitar a hipótese nula, isto é, medida F de ambas as abordagens é diferente se comparadas ao mapeamento de um especialista. Porém, como existe diferença entre as médias das abordagens, sugere-se analisar melhor esta relação, conforme a Tabela 5.12.

Tabela 5.12 – Análise da média descritiva das abordagens.

Abordagem	**Média**
Automática	0,9428
Manual	0,7952

Comparando os dados apresentados na Tabela 5.12, observa-se que a média descritiva da medida F na amostra automática é maior do que na manual. Esta evidência é pontual para este experimento e em outras execuções há a possibilidade estatística desta relação mudar.

5.2.4. Interpretação

O ponto fundamental para a identificação do mapeamento automático é a habilidade de identificar quais conceitos do domínio estão presentes em uma especificação de software. Na especificação textual, foram considerados durante o experimento todos os problemas inerentes da língua natural, como ambiguidade, termos implícitos, pronomes, etc. Este mapeamento não é trivial e idealmente deveria ser executado por um especialista do domínio, discernindo e relacionando a perspectiva conceitual das demais perspectivas de um sistema de informação (no caso deste estudo, a perspectiva comportamental foi avaliada).

Este experimento foi conduzido com o intuito de avaliar o quão próximo é o resultado obtido pela análise de similaridade automática e manual entre conceitos do domínio e artefatos de software. Neste experimento, buscou-se o estado da prática em diversas de fábrica de software no Brasil, utilizando documentação especificada na língua inglesa e avaliada por pessoas não nativas neste idioma. Para apoiar os indivíduos na

execução de suas tarefas, foi permitido o uso de dicionários ou consulta exclusiva ao condutor do experimento com relação aos termos da linguagem.

Uma das evidências encontradas neste estudo foi à perda de revocação da abordagem manual na descrição a qual os conceitos não estavam explicitamente definidos. Como exemplo, tem-se o Caso de Uso 04, na qual foram identificados apenas 2 conceitos em um universo de 8. Existe a descrição neste caso de uso de termos sinônimos a alguns conceitos, e esta sutileza não foi percebida por alguns dos participantes.

Da mesma forma, algumas características foram evidenciadas para o caso automático, como a diminuição da precisão pela identificação de falsos positivos. Caso existam termos não relacionados a especificações, referenciados através de sinônimos ou explicações descontextualizadas, este mapeamento é identificado pela ferramenta. Durante a especificação da análise de similaridade, não foi estabelecido um ponto de corte para avaliar falsos positivos. Este ponto de corte foi implementado especificamente no modelo probabilístico ao final do processo completo de análise de impacto. O fenômeno de falsos positivos também foi identificado no caso do mapeamento manual, onde indivíduos inferiram relacionamentos que efetivamente não fazem parte do contexto da especificação.

Como resultado do estudo experimental, foi possível evidenciar que, comparado o mapeamento automático com o efetuado por um especialista, ambas as abordagens geram amostras com diferença estatística entre suas médias. Através da análise descritiva, conseguiu-se identificar que a média geral da abordagem automática possui uma medida F maior que a média da abordagem manual. Considerando a média parcial da abordagem automática e manual, verifica-se que a medida F obtida pela ferramenta é maior em 83% das especificações consideradas, isto é, em 5 das 6 especificações avaliadas, o que traz bons indícios da aplicabilidade da análise em consideração.

5.3. Experimento sobre Esforço associado à Análise de Impacto

5.3.1. Definição

A abordagem GQM também foi utilizada para a definição do estudo. O objetivo foi avaliar o esforço necessário para analisar e implementar uma mudança em uma aplicação utilizando ou não a ferramenta SEmantics. Este objetivo derivou a seguinte questão: “*o*

esforço para a implementação de uma mudança utilizando µaut (abordagem automatizada utilizando SEmantics) é o mesmo que utilizando µman (abordagem manual)?". A métrica adotada foi o tempo gasto em minutos em cada atividade.

O objetivo de estudo foi definido como:

Comparar o tempo necessário para analisar e implementar mudanças,

Com o propósito de avaliar o desempenho da ferramenta SEmantics,

Com foco no esforço,

Sob o ponto de vista de um programador de software,

No contexto de uma manutenção evolutiva.

5.3.2. Planejamento e Execução

O experimento foi conduzido em uma universidade utilizando um ambiente controlado e *off-line*. A população definida para o experimento foi formada por graduandos de computação finalizando o curso de linguagem de programação orientada a objetos (Java). A amostra da população foi não probabilística, escolhida por quota e conveniência. A questão previamente definida originou a seguinte Hipótese Nula (**H_0**): o esforço necessário para a implementação de uma mudança em um software utilizando a ferramenta SEmantics é igual ao esforço necessário utilizando uma abordagem manual. O esforço foi avaliado como sendo o tempo gasto em minutos para implementar a mudança em cada abordagem, isto é, a diferença entre o tempo final e o tempo inicial, onde:

I. Tµaut: representa a variação de tempo gasto em minutos para implementação da mudança utilizando a ferramenta SEmantics;

II. Tµman: representa a variação de tempo gasto em minutos para implementação da mudança utilizando a abordagem manual.

H_0: Tµaut = Tµman

H_1: Tµman > Tµaut

H_2: Tµaut > Tµman

A instrumentação utilizou uma aplicação Web na linguagem Java disponível em JavaFree.org [Jfr12], responsável por gerenciar locações de veículos. Trata-se de uma aplicação composta por 105 classes e estruturada em uma arquitetura MVC que faz uso de *servlets*, JSPs, DAOs e classes de controle. A aplicação é responsável por gerenciar

dados como veículos, locação de veículos, clientes, estados, cidades, usuários do sistema, bem como marcas, categorias e modelos de veículos. A ontologia utilizada para ferramenta SEmantics foi extraída do modelo de domínio disponível pela aplicação. Para esta aplicação, foram definidos os seguintes requisitos de mudanças:

- Quando cadastrar ou alterar o modelo de um veículo, garantir que a descrição não tenha mais que 10 caracteres.
- Quando inserir um usuário ou alterar um usuário, validar que a senha tenha mais que 6 caracteres.

As variáveis independentes foram μaut e μman e a variável dependente foi o tempo gasto por cada participante para compreender e implementar a mudança. O projeto experimental utilizado foi de um fator (esforço) com dois tratamentos (μaut e μman). Para a abordagem μaut, a análise de impacto e implementação da mudança foi executada utilizando o IDE Eclipse com o *plugin* SEmantics. Para a abordagem μman, foi fornecido apenas o Eclipse. Para coleta dos dados, os participantes preencheram um formulário registrando a hora de início e término de cada atividade. Após registrado o tempo, os participantes apresentaram o software executando com a mudança requisitada para o condutor do experimento, validando assim a implementação. A instrumentação completa do experimento é apresentada no Apêndice H.

O experimento possuiu 13 participantes, cada um executando duas requisições de mudanças (RdM), resultando em duas amostras de 13 pontos. Os participantes receberam toda a infraestrutura do software pronta para execução da análise de impacto e implementação da mudança. A infraestrutura incluiu o software executando no ambiente de desenvolvimento (Eclipse) configurado com o servidor de banco de dados (HSQLDB) e de aplicação (Tomcat). Os participantes foram divididos por técnicas e fornecido treinamentos específicos (ferramenta SEmantics e análise de impacto manual). Foi solicitado que cada participante interpretasse e implementasse cada mudança, aplicando ou a técnica μaut ou μman, definida a priori de forma aleatória.

5.3.3. Análise

Para este experimento, a variável independente foi caracterizada na escala nominal (abordagem) e a dependente na escala razão (esforço). O nível de significância adotado foi de 5%.

A análise foi estruturada a partir das duas RdMs: durante a primeira, os participantes não possuíam nenhum conhecimento sobre a arquitetura da aplicação, enquanto durante a segunda, os mesmos já possuíam familiaridade na arquitetura do software devido a primeira implementação (ocasionando uma diferença significativa no esforço). A execução do experimento produziu os dados apresentados na Tabela 5.13.

Tabela 5.13 – Conjunto de Dados da Execução do Experimento.

1ª Requisição de Mudança (1ª RdM)			**2ª Requisição de Mudança (2ª RdM)**		
Abordagem	Particip.	Esforço (min.)	Abordagem	Particip.	Esforço (min.)
µman	P01	0:30:00	µman	P01	0:07:00
µman	P02	0:34:00	µman	P02	0:14:00
µman	P03	0:52:00	µman	P03	0:09:00
µman	P04	0:40:00	µman	P04	0:08:00
µman	P05	0:47:00	µman	P05	0:15:00
µman	P06	0:15:00	µman	P06	0:08:00
µman	P07	0:30:00	µman	P07	0:15:00
µman	P08	0:45:00	µman	P08	0:07:00
µman	P09	0:36:00	µaut	P09	0:10:00
µaut	P10	0:08:00	µaut	P10	0:05:00
µaut	P11	0:33:00	µaut	P11	0:06:00
µaut	P12	0:15:00	µaut	P12	0:03:00
µaut	P13	0:22:00	µaut	P13	0:05:00

A análise inicial avaliou a distribuição dos dados através dos gráficos de linhas, conforme Figura 5.7, e *boxplots*, conforme Figura 5.8. Como os *outliers* na Figura 5.8 foram considerados moderados, nenhum participante foi retirado da amostra.

O próximo passo foi avaliar a normalidade da amostra, considerando as duas hipóteses definidas no experimento anterior. A Tabela 5.14 apresenta os resultados do Teste de Shapiro-Wilk.

Tabela 5.14 – Teste de normalidade Shapiro-Wilk.

RdM	**Variável**	**Abordagem**	**Estatística**	**Grau de Liberdade**	**Significância**
1ª RdM	Esforço	µaut	0,987	4	0,942
		µman	0,962	9	0,821
2ª RdM	Esforço	µaut	0,894	5	0,376
		µman	0,782	8	0,078

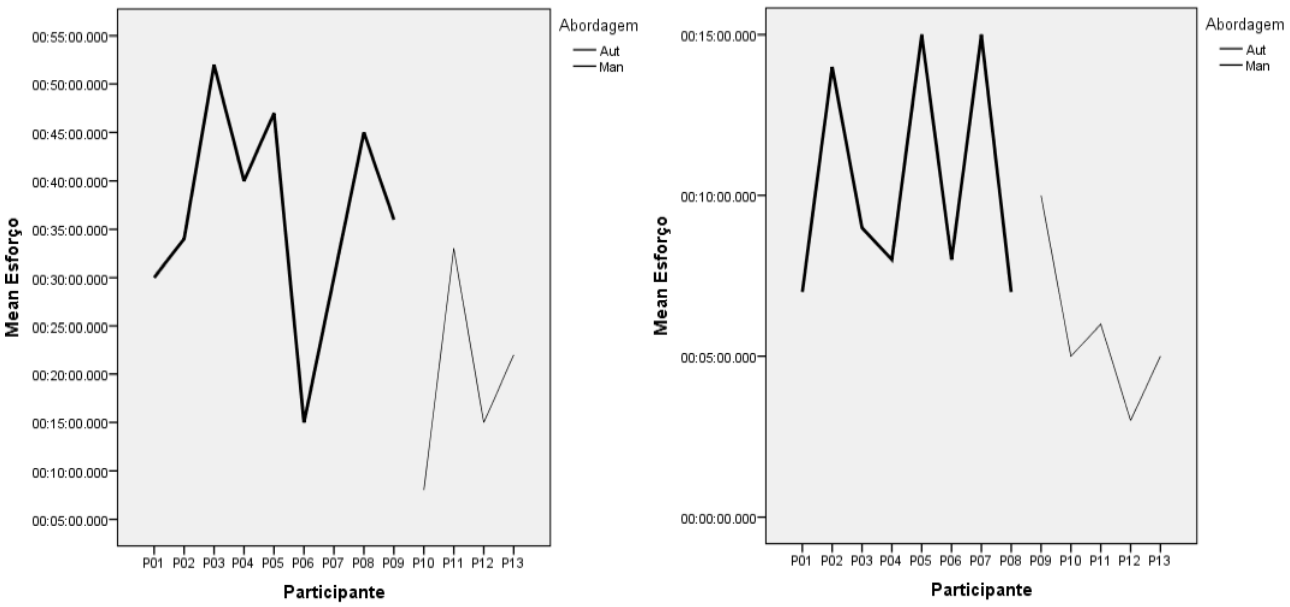


Figura 5.7 – Gráficos de Linhas da 1ª RdM (esq) e 2ª RdM (dir).

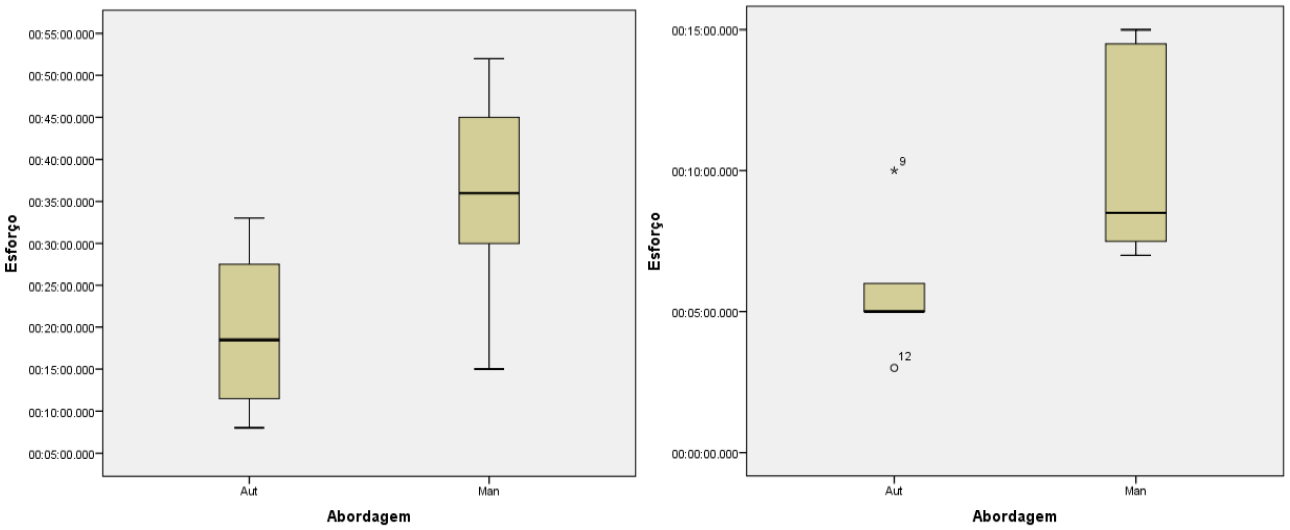


Figura 5.8 – Gráficos de dispersão da 1ª RdM (esq) e 2ª RdM (dir).

Com base na Tabela 5.14, observa-se que as significâncias dos dados do teste de Shapiro-Wilk são superiores nas duas RdMs considerando o nível de significância definido (0,05). Com esta informação, não se pode rejeitar a hipótese nula sobre a distribuição da normalidade. O segundo requisito para o teste paramétrico requer a análise da homocedasticidade. Para tanto, foram utilizadas as mesmas hipóteses definidas anteriormente para o Teste de Levene. A Tabela 5.15 apresenta os resultados deste teste.

Tabela 5.15 – Teste de Levene para igualdade das variâncias sobre o esforço.

RdM		Significância
1ª RdM	Assumindo variâncias iguais	0,919
	Não assumindo variâncias iguais	
2ª RdM	Assumindo variâncias iguais	0,086
	Não assumindo variâncias iguais	

Com base na Tabela 5.15, verifica-se que os níveis de significância para variâncias iguais na 1ª RdM (0,919) e 2ª RdM (0,086) são superiores ao nível de significância definido (0,05), viabilizando a utilização do teste paramétrico para duas amostras independentes.

Com base na declaração das hipóteses, têm-se:

H_0: Tμaut = Tμman

H_1: Tμman > Tμaut

H_2: Tμaut > Tμman

O Teste T para duas amostras independentes foi aplicado neste contexto e o resultado está apresentado na Tabela 5.16.

Tabela 5.16 – Teste T para esforço agrupado μman e μaut e por RdMs

RdM		**T**	**Grau de Liberdade**	**Significância (bicaudal)**
1ª RdM	Assumindo variâncias iguais	-2,582	11	0,026
2ª RdM	Assumindo variâncias iguais	-2,443	11	0,033

Observando a Tabela 5.16, percebe-se que as significâncias (-0,026 e -0,033) são inferiores ao *p-value* definido (0,05), permitindo assim rejeitar **H_0** (Tμaut = Tμman). Isso significa que as médias de esforço das duas amostras (μaut e μman) são estatisticamente diferentes para as duas RdMs.

A Tabela 5.16 também apresenta que os valores de t_0 (= -2,582) na 1ª RdM e t_0 (= -2,443) na 2ª RdM são inferiores ao valor de $t_{\alpha,\ n+m-2}$ (= 2,20) para o grau de liberdade 11. Como $t_0 < t_{\alpha,\ n+m-2}$ nas duas RdMs, então não se consegue rejeitar a hipótese nula a um nível de significância de 5% em favor de **H_1**: Tμman > Tμaut para os dois requisitos de mudança.

Para **H_2** (Tμaut > Tμman), foi aplicado Teste T cujo o resultado está descrito na Tabela 5.17.

Tabela 5.17 – Teste T para esforço agrupado μaut e μman e por RdMs

RdM		T	Grau de Liberdade	Significância (bicaudal)
1ª RdM	Assumindo variâncias iguais	2,582	11	0,026
2ª RdM	Assumindo variâncias iguais	2,443	11	0,033

Com base na Tabela 5.17, obtêm-se que os valores de t_0 (=2,582) na 1ª RdM e t_0 (=2,443) na 2ª RdM são maiores que $t_{\alpha, n+m-2}$ (= 2,20). Como $t_0 > t_{\alpha, n+m-2}$ nas duas RdMs, então se consegue rejeitar a hipótese nula a um nível de significância de 5% em favor de **H_2**: Tμaut > Tμman para os dois requisitos de mudança.

Pelas análises apresentadas, pode-se concluir que existe diferença estatisticamente significativa com relação ao esforço necessário em analisar o impacto e implementar uma mudança utilizando a ferramenta SEmantics comparada a uma abordagem manual. O esforço utilizando a abordagem manual é maior que o esforço utilizando a ferramenta SEmantics. Isso representa que a ferramenta, no contexto deste experimento, ajudou os participantes a analisar o impacto e implementar a mudança se comparada a uma abordagem manual.

5.3.4. Interpretação

O terceiro experimento teve como objetivo avaliar a eficiência do modelo de análise de impacto em uma requisição de mudança, considerando desde a interpretação do requisito até sua efetiva implementação. Para tanto, foi identificado um software desenvolvido por terceiros e definido duas alterações.

Durante a execução da primeira modificação, os participantes não possuíam nenhum conhecimento sobre o software objeto deste estudo nem de sua arquitetura. Quando foi solicitada a segunda alteração, o conhecimento adquirido pela primeira execução (fator experiência) fez com que o esforço fosse consideravelmente reduzido em ambas as abordagens. Cada requisição de mudança foi avaliada de forma independente para caracterizar especificamente esses dois momentos.

A hipótese de que o tempo necessário para a implementação das mudanças é o mesmo nas duas abordagens foi rejeitada utilizando o teste paramétrico T. Este teste

deixa claro que existe diferença significativa entre analisar e implementar uma mudança utilizando ou não a ferramenta SEmantics. Em seguida, foram avaliadas duas hipóteses alternativas para identificar qual abordagem requer maior esforço para ser executada e, pela análise dos dados, foi evidenciado que a abordagem que utilizou a ferramenta SEmantics levou menos tempo que a manual/tradicional.

Observando a distribuição dos dados, percebe-se que a dispersão diminuiu significativamente para a abordagem automática conforme os participantes adquiriam conhecimento sobre a arquitetura do software. Durante o primeiro requisito de mudança, quando os participantes não possuíam nenhum conhecimento, houve uma dispersão significativa de esforço. Este fato não ocorreu para a segunda mudança, gerando inclusive a uma sobreposição quase nula de valores entre as amostras, a exceção de um *outlier*. Com isso, evidencia-se que quanto maior o conhecimento do software, menor a dispersão dos dados, gerando conjuntos praticamente disjuntos entre a abordagem manual e automática. Este fato sugere que a ferramenta SEmantics tende a tirar a variabilidade humana do processo de análise de impacto, visto que o esforço não variou muito entre os participantes do experimento a medida que eles vão adquirindo conhecimento sobre o código fonte do software.

5.4. Considerações Finais

Este capítulo apresentou uma avaliação quantitativa do modelo de análise de impacto em código fonte utilizando ontologias através de três experimentos controlados. Estes experimentos exploraram aspectos relativos a revocação e precisão da análise de impacto resultante da ferramenta SEmantics e da análise de similaridade que identifica automaticamente conceitos do domínio a partir de artefatos de software. Também foi avaliado o esforço referente a analisar e implementar mudanças utilizando ou não a ferramenta SEmantics. O objetivo geral foi avaliar a aplicabilidade e relevância da automação do modelo de análise de impacto, considerando diferentes aspectos quando comparada a atividade equivalente desempenhada por humanos.

Durante o experimento que analisou a precisão e revocação de três diferentes requisições de mudanças, evidenciou-se estatisticamente que a medida F é maior utilizando a ferramenta SEmantics do que a obtida manualmente por engenheiros de software. As estratégias manuais são dependentes de especificação estruturada e, conforme descrito por Jönsson [Jon07], correm o risco de serem menos precisas na

predição de impacto. Este fenômeno foi evidenciado neste experimento onde, para as três requisições mudanças analisadas, em nenhum caso a abordagem manual teve melhor desempenho com relação a precisão ou revocação. Vale ressaltar que a ferramenta SEmantics possui um melhor desempenho na predição de impacto quando realizada a referência adequada aos conceitos e propriedades do domínio para identificar estruturas do código que sejam equivalentes no escopo da aplicação. Isto requer que haja uma coerência semântica entre a definição da ontologia e da estrutura da aplicação, e que as estruturas corretas sejam evidenciadas pela requisição de mudança.

A análise de similaridade, que corresponde a um processo chave para a automação do modelo de análise de impacto, também foi especificamente avaliada. Este experimento utilizou a medida F para comparar a precisão e revocação na identificação automática e manual de conceitos do domínio associados a artefatos de software. Uma vez associados os conceitos à estrutura do software, esta análise de similaridade é utilizada para definir as métricas de probabilidade aplicadas para avaliar a relevância de cada associação, como TFIDF e DC. Este é um processo crítico do modelo que foi julgado como relevante para uma análise minuciosa através de experimentos controlados.

Com relação ao segundo experimento, o resultado de sua execução definiu um contexto diferente do primeiro, onde a amostra não apresentou uma distribuição normal, impossibilitando a utilização do Teste T. O teste Mann-Whitney foi à escolha definida para avaliação das hipóteses. Este teste apenas informa se dois grupos independentes procedem da mesma população. O resultado foi que existe diferença entre as medidas F associadas a análise de similaridade manual e automatizada. Neste contexto, não foi possível avaliar estatisticamente as hipóteses alternativas e se optou por comparar a análise descritiva das médias. Após comparar as médias, verificou-se que a medida F na análise de similaridade automática é aproximadamente 15% maior que a análise manual e que medida F obtida pela ferramenta é maior em 83% das especificações consideradas. Analisando cada caso, em todas as amostras, a revocação foi maior na análise de similaridade automática e a precisão em 83% dos casos.

O terceiro experimento teve como objetivo caracterizar o esforço necessário para interpretar e implementar uma mudança no código fonte considerando dois momentos: primeiro, sem o conhecimento da arquitetura do software, e o segundo, com este conhecimento. Para tanto, foram consideradas duas requisições de mudanças implementadas por programadores com conhecimento equivalente na linguagem do programa em questão. Através do experimento, foi possível provar que existe diferença

estatística entre as abordagens, referente ao tempo gasto para realizar a manutenção no software, e rejeitar a hipótese de que é necessário maior esforço para implementar a mudança utilizando a ferramenta SEmantics. Isto significa que, para as duas requisições, o tempo necessário para implementar a mudança é maior utilizando a abordagem manual.

Considerando a análise descritiva do terceiro experimento, identifica-se que a média de tempo necessária para implementar o primeiro requisito de mudança, utilizando a abordagem manual, ficou em aproximadamente 36 minutos, enquanto a automatizada em 19 minutos. Esta diferença equivale a quase o dobro do tempo se não for utilizada a ferramenta SEmantics. Esta variação foi mantida durante o segundo requisito de mudança, que obteve uma média em torno de 10 minutos para a abordagem manual e 5 para a automatizada. Percebe-se que a proporção de 50% para implementar a mudança foi mantida. Essa diferença é significativa e, mesmo em um contexto no qual o programador já possui conhecimento da arquitetura do software, o benefício na utilização da ferramenta com relação ao esforço se mostrou igualmente interessante, reduzindo o tempo, em média, a metade.

É importante evidenciar que os resultados obtidos através destes experimentos não são generalizáveis para qualquer análise de impacto em software. Os resultados foram específicos para seus respectivos escopos, incluindo a limitação no tamanho de cada amostra. Para aumentar o conhecimento sobre precisão e revocação em diferentes contextos, definindo a validade da experimentação, sugere-se replicações do experimento em modelos distintos e em amostras mais significativas (tanto em quantidade quanto qualidade). Generalizando o experimento, possibilita-se a extração de novas informações sobre a proposta em diferentes perspectivas no contexto de manutenção de software.

6. CONCLUSÕES

A relevância da análise de impacto durante a manutenção de software é notória. Mudanças são comuns em um cenário evolutivo, no qual novas funcionalidades são adicionadas ou adaptadas. Para tanto, modelos para analisar impacto se tornam fundamentais.

Conforme revisão apresentada, percebe-se que as propostas de automação de análise de impacto em código fonte se baseiam em estratégias manuais e automatizadas. Dentre as automatizadas, os modelos que se destacam utilizam a análise léxica do código fonte associadas a modelos de recuperação de informação ou pela própria análise de dependência das estruturas do código, o que gera algumas limitações sob o ponto de vista de negócios. Estas limitações estão associadas a distância que existe entre o problema, definido pelo conhecimento de um especialista do domínio, e a solução, definida pela equipe de desenvolvimento, que irá gerar em sua última instância o código fonte da aplicação.

Neste contexto, o modelo de análise de impacto proposto tem como objetivo automatizar o processo de recuperação de estruturas do código fonte relevantes dada uma solicitação de mudança. Este modelo foi organizado em dois submodelos: de rastreabilidade e probabilístico, ambos apoiados por ontologias. A motivação em utilizar ontologias reside na ideia de que o contexto semântico da aplicação, e não apenas a análise das estruturas estáticas do código, pode gerar uma análise mais apurada dos impactos resultantes de determinada solicitação de mudança.

Foi formulado um tópico genérico de pesquisa dentro do problema identificado e desenvolvidos estudos em abrangência, tanto sobre análise de impacto quanto seu relacionamento com ontologias e recuperação de informação. Uma vez desenvolvida a fundamentação da área, foi realizada uma revisão sistemática para verificar o estado da arte com relação ao foco desta pesquisa, não identificando explicitamente nenhuma proposta completamente equivalente. Logo em seguida, foi apresentado o modelo de análise de impacto que é constituído por um modelo de rastreabilidade e probabilístico.

O modelo de rastreabilidade permite a associação de conceitos do domínio com o código fonte da aplicação através da população da ontologia de domínio com elos de rastreabilidade. Devido à granularidade específica dessas associações em nível de conceitos e suas propriedades, foi identificada a necessidade de automatizar o processo de geração de elos de rastreabilidade. Para tanto, foi desenvolvido um processo

automatizado de análise de similaridade entre conceitos e artefatos de software. Este processo é realizado considerando duas perspectiva: a semântica, responsável pela categorização e normalização, e a léxica, responsável pela comparação.

O modelo probabilístico foi desenvolvido utilizando Redes de Crenças Bayesianas, que pondera os elos de rastreabilidade conforme as estruturas da ontologia identificadas em determinado requisito de mudança. Para tanto, são calculadas probabilidade independentes, utilizando o algoritmo de classificação PageRank, e dependentes, realizando análise de frequência e cálculo de dependência conceitual, que considera a sobreposição de grafos de chamada no código fonte com a distância conceitual na ontologia.

O modelo de análise de impacto foi implementando como um *plugin* do Eclipse e sua arquitetura foi apresentada em detalhes. Após o desenvolvimento da ferramenta, foram realizados três experimentos controlados para avaliar a precisão, revocação e esforço do modelo de análise de impacto com relação a estratégias convencionais.

A execução dos dois primeiros experimentos evidenciou que tanto a análise de impacto quanto seu subprocesso de análise de similaridade, responsável pela automação de toda a proposta, obtiveram um desempenho bastante interessante considerando o critério de precisão e revocação. Foi verificada estatisticamente que a medida F associada a análise de impacto é maior no modelo proposto nesta tese, com relação a recuperação de código relevante, quando comparada a abordagem manual de análise de impacto. A execução do terceiro experimento evidenciou estatisticamente que o esforço para implementar uma mudança em um software é maior utilizando a abordagem manual se comparada a abordagem utilizando a ferramenta SEmantics, tanto com o conhecimento prévio da arquitetura do software quanto sem este conhecimento.

Com relação a metodologia, observa-se que a questão de pesquisa definida para esta tese, que corresponde em “Como melhorar a recuperação de estruturas de código fonte relevantes a partir de uma solicitação de mudança comparada a abordagens existentes de análise de impacto?”, foi respondida integralmente ao longo desde trabalho. Este trabalho não só apresentou em detalhes o modelo para automatizar a análise de impacto, mas também o avaliou em duas perspectivas: viabilidade técnica e eficiência. A primeira avaliação considerou um protótipo funcional enquanto a segunda, três experimentos controlados. Os experimentos exploraram a precisão e revocação das estruturas de código recuperadas, comparando-as com estratégias convencionais de análise de impacto, bem como o esforço necessário para interpretar e implementar uma

mudança com ou sem a ferramenta SEmantics, que implementa o modelo de análise de impacto proposto.

Com relação ao objetivo geral de desenvolver um modelo para identificar estruturas do código fonte impactadas por solicitações de mudanças, utilizando uma perspectiva semântica, verifica-se que este objetivo também foi atendido completamente conforme descrito no Capítulo 3. A seguir, serão analisados cada objetivo específico definido nesta pesquisa:

1. *evoluir o modelo de análise de rastreabilidade utilizando ontologias apresentado em [Nol07a];*

    - o modelo de rastreabilidade desenvolvido em [Nol07a] evoluiu para um contexto mais abrangente que a rastreabilidade entre conceitos da ontologia e elementos UML atrelados ao *Rational Unified Process*. A evolução do modelo foi apresentada em detalhes na Seção 3.2, descrevendo as estruturas necessárias para a indexação do código fonte, bem como o relacionamento entre classes, métodos e atributos do código com conceitos e propriedades da ontologia.

2. *desenvolver um processo para população da ontologia pela automação do mapeamento de estruturas do código fonte e conceitos do domínio;*

    - foi desenvolvido um processo para automatizar a análise de similaridade, relacionando estruturas de baixo nível do código fonte com conceitos e propriedades do domínio através da população automática de uma ontologia de rastreabilidade. Esta população utilizou um conjunto de métricas de processamento de linguagem natural, que incluem categorização, normalização e comparação através de uma análise léxica e semântica.

3. *desenvolver um modelo de análise de probabilidade utilizando modelos e técnicas de recuperação de informação e grafos de chamadas;*

    - a Seção 3.3. apresenta o modelo probabilístico utilizado para ponderar os elos de rastreabilidade entre o domínio do sistema e sua implementação lógica. Este modelo foi desenvolvido utilizando Redes de Crenças Bayesianas, através da definição de nodos (e estados) e arestas (com as probabilidades relativas). As probabilidades independentes foram definidas utilizando o algoritmo de classificação

PageRank para os conceitos do domínio e classes do código fonte. As probabilidades condicionais utilizaram a análise de frequência (TFIDF) e dependência conceitual (sobreposição do grafo de chamada do código fonte com a distância dos conceitos na ontologia).

4. *desenvolver um protótipo funcional como avaliação da viabilidade do modelo;*

   - O modelo de análise de impacto foi implementado integralmente conforme descrito no Capítulo 4. A solução, por se tratar de uma análise no código fonte, incluiu a criação de um *plugin* do Eclipse chamado SEmantics. Através desse *plugin*, o usuário pode selecionar o projeto Java, a ontologia e descrever a mudança. Ao final, as estruturas do código fonte (classes, atributos e métodos) são ponderadas pelo modelo de análise de impacto.

5. *avaliar empiricamente a proposta utilizando experimentos.*

   - Foram realizados três experimentos controlados visando avaliar a precisão e revocação dos elementos recuperados, bem como o esforço necessário para analisar e implementar uma mudança. O primeiro experimento avaliou o modelo de análise de impacto como um todo, enquanto o segundo avaliou especificamente a análise de similaridade, que representa uma parte central deste modelo, responsável pela automação na identificação de estruturas do código associadas a conceitos do domínio. O terceiro foi responsável por avaliar o esforço durante a manutenção do software, aplicando o modelo de análise de impacto proposto.

6.1. Contribuição da Tese

A principal contribuição da tese proposta é o desenvolvimento de um modelo para análise de impacto e sua avaliação empírica em um típico cenário de desenvolvimento, considerando os desafios existentes relacionados à evolução de software e gerenciamento de mudanças. Hass em [Has02] evidencia que qualquer organização que inicia um processo de gerenciamento de mudanças e configuração deverá focar na

perspectiva do processo para entender uma melhoria, e obviamente a análise de impacto faz parte desta análise.

Como contribuições específicas desta tese, destacam-se:

- a identificação e análise dos impactos em estruturas do código fonte a partir de requisições de mudanças;
- o desenvolvimento de um modelo de análise de rastreabilidade utilizando ontologias capaz de relacionar conceitos do domínio com o código fonte da aplicação. Este modelo é flexível o suficiente para relacionar conceitos com diferentes artefatos de software, tais como código fonte, diagramas, requisitos, especificação técnica, notas de reuniões, etc.;
- o desenvolvimento de um modelo probabilístico capaz de identificar unidades de código fonte relevantes para determinado requisito de mudança, utilizando como base o modelo de rastreabilidade apoiado por ontologias;
- o desenvolvimento da métrica de Dependência Conceitual, que equivale ponderar se a distância entre os conceitos do domínio é equivalente a distância entre as classes do código fonte;
- o desenvolvimento de um protótipo funcional junto a uma ferramenta integrada de desenvolvimento de software;
- a avaliação empírica dos resultados do modelo de análise de impacto.

Ainda como contribuições desta pesquisa, a Tabela 5.1 descreve as produções direta ou indiretamente relacionadas com algum aspecto da tese.

Tabela 6.1 – Publicações relacionadas a tese.

Referência	Classificação	Estado	*Descrição*
[NOL12b]	B2	Em análise	Noll, R.P.; Blois, M.R., Bastos, R. M.“Ontology Population for Ontology-Driven Traceability”. In: International Journal of Software Engineering and Knowledge Engineering (IJSEKE).
[NOL12a]	B2	Aprovada, não publicada	Noll, R.P.; Blois, M.R., Bastos, R. M.“Ontology Population for Ontology-Driven Traceability”. In: International Conference on Enterprise Information (ICEIS), Poland, July, 2012.
[NOL11]	-	Publicada	Noll, R. P. “Um Modelo para a Análise de Impacto em Código Fonte usando Ontologias”. Proposta de Tese, Faculdade de Informática - PUCRS, 2011, 101p.
[NOL09]	B2	Publicada	Noll, R.P.; Blois, M.R. “Concepts-Based Traceability: Using Experiments to Evaluate Traceability Techniques”. In: International Conference on Enterprise Information (ICEIS), Milan/Italy, May, 2009.

[NOL08]	B4	Publicada	Noll, Rodrigo Perozzo ; Blois, M. . Usando experimentos para avaliar o mapeamento automático entre conceitos do domínio e especificação de Software. In: V Experimental Software Engineering Latin American Workshop, 2008, Salvador. 5th Experimental Software Engineering Latin American Workshop, 2008. v. 5.
[NOL07d]	B2	Publicada	Noll, R. P.; Saccol, D. B.; Edelweiss, N. Uma proposta para análise de similaridade entre documentos XML e ontologias em OWL. In: Simpósio Brasileiro de Banco de Dados; 2007. Proceedings... João Pessoa: 2007.
[NOL07c]	B3	Publicada	Noll, R.; Blois, M. “Enhancing Traceability using Ontologies”. SAC2007 - The 22nd Annual ACM Symposium on Applied Computing. Seoul, Korea, March 11 - 15, 2007.
[NOL07b]	B2	Publicada	Noll, R.; Blois, M. “Ontological Traceability over the Unified Process”. ECBS2007 - 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems. Arizona, USA, March 26 - 29, 2007.
[NOL07a]	-	Publicada	Noll, R. P. “Rastreabilidade Ontológica sobre o Processo Unificado”. Dissertação de mestrado, Faculdade de Informática - PUCRS, 2007, 116 p.

6.2. Limitações da Proposta e Trabalhos Futuros

Como todo processo de pesquisa, foram identificadas algumas restrições. Estas limitações são derivadas tanto da metodologia quanto modelos utilizados.

Apesar da necessidade do desenvolvimento de uma ontologia de domínio, não é foco primário desta proposta fornecer os meios nem os processos de construção dessa ontologia. Para a construção da ontologia, engenheiros do conhecimento poderão utilizar as guias apresentadas no Apêndice C. É importante destacar que os profissionais que utilizarão o modelo de análise de impacto não necessitam de conhecimento sobre a engenharia de ontologias, eles apenas a utilizam como insumo para identificação de software relevantes.

Não é escopo deste trabalho a manutenção dos elos de rastreabilidade, que caracteriza a evolução de ontologias. Uma vez criado os elos, os mesmos serão utilizados para recuperação de estruturas de código. A definição dos elos de rastreabilidade ocorre em tempo de execução, sempre que uma requisição de mudança é solicitada. Estes elos não são armazenados fisicamente. Qualquer mudança no contexto da aplicação ou no domínio requer nova análise de impacto. Como esse mapeamento em geral é feito automaticamente, o esforço para gerar novo modelo de análise de impacto não é crítico.

Por último, a linguagem de programação utilizada para análise de impacto no código fonte é Java, devido as bibliotecas utilizadas para avaliar a viabilidade da proposta.

Como trabalhos futuros, sugere-se evoluir o modelo de análise de impacto com a manutenção e persistência física dos elos de rastreabilidade, bem como sua manutenção em tempo de execução. Sugere-se também considerar diferentes linguagens de programação e IDEs. Por último, é sugerido também evoluir o modelo para analisar o impacto em diferentes artefatos de software, não apenas no código fonte.

REFERÊNCIAS BIBLIOGRÁFICAS

[Aji95] Ajila, S. "Software maintenance: an approach to impact analysis of objects change". Software - Practice and Experience, John Wiley and Sons Ltd, 1995, pp. 155-181.

[Ama12] Amazon.com. “Amazon”. Capturado em: http://www.amazon.com/, Maio 2012.

[Ant00] Antoniol,G.; Canfora, G. ;Casazza, G.; Lucia, A. D. "Identifying the starting impact set of a maintenance request". In: Proceedings of 4th European Conference on Software Maintenance and Reengineering, 2000, pp. 227–230

[Ara06] Araújo, M. A.; Barros, M.; Travassos, G.; Murta, L. "Métodos Estatísticos aplicados em Engenharia de Software Experimental". XXI SBBD - XX SBES, 2006, 35p.

[Arg12] Tigris.org. "ArgoUML" . Capturado em: http://argouml.tigris.org/. Acesso em Maio 2012.

[Arn93] Arnold, R.S.; Bohner, S. A. “Impact analysis-Towards a framework for comparison”. In: Conference on Software Maintenance, 1993, pp. 292-301.

[Arn96] Arnold, R. S.; Bohner, S. “Software Change Impact Analysis”, IEEE Computer Society Press, 1996, 392p.

[Arr01] Arrington, C.T. “Enterprise Java with UML”, Wiley, 2001, 512p.

[Ary11] Aryani, A.; Perin, F.; Lungu, M.; Mahmood, A. N.; Nierstrasz, O. "Can we predict dependencies using domain information ?". In: 18th Working Conference on Reverse Engineering, 2011, pp. 55-64.

[Ass09] Assawamekin, N.; Sunetnanta, T.; Pluempitiwiriyawej, C. “Deriving Traceability Relationships of Multiperspective Software Artifacts from Ontology Matching”. In: 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009, pp. 549-554.

[Bae99] Baeza-Yates, R.; Ribeiro, B. N. "Modern Information Retrieval". Addison-Wesley Longman Publishing Co., Inc, 1999, 513p.

[Bas94] Basili, V.R.; Caldiera, G.; RombacH, H. D. "The Goal Question Metric Approach". Encyclopedia of Software Engineering. Wiley, 1994, 10p.

[Bat07] Batanov, D.; Vongdoiwang, W. “Using Ontologies to Create Object Model for Object-Oriented Software Engineering”. Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems, Springer US, 2007, pp. 461-487.

[Ber79] Bersoff, E. H.; Henderson, V. D., Siegel, S. G. "Software Configuration Management: A Tutorial". Computer, 1979, pp. 6-14.

[Bia00] Bianchi, A; Fasolino, A. R.; Visaggio, G. “An exploratory case study of the maintenance effectiveness of traceability models”. In: Proceedings 8th International Workshop on Program Comprehension, 2000, pp. 149-158.

[Bis04] Bisquerra, R.; Sarriera, J. C.; Martínez, F. "Introdução a estatística. Enfoque informático com o pacote estatístico SPSS". Artmed Editora, Porto Alegre, 2004, 315p.

[Boh96] Bohner, S. A. "Impact Analysis in the Software Change Process: A Year 2000 Perspective". In: Proceedings of International Conference on Software Maintenance, 1996, pp. 42–51.

[Bor11] Borg, M. "Do Better IR Tools Improve the Accuracy of Engineers Traceability Recovery ?" MALETS, 2011, pp. 27-34.

[Bor97] Borst, W.N. “Construction of Engineering Ontologies for Knowledge Sharing and Reuse”. PhD Thesis, University of Tweenty, 1997, 227p.

[Bri98] Brin, S.; Page, L. "The anatomy of a large-scale hypertextual Web search engine". In: Proceedings XVII Conference on World Wide Web, 1998, pp. 107-117.

[Buc05 Buckner, J.; Buchta, J.; Petrenko, V.; Rajlich, M. “JRipples: A Tool for Incremental Software Change”, IEEE International Workshop on Program Comprehension, 2005, pp. 149-152.

[Can05] Canfora, G.; Cerulo, L. "Impact Analysis by Mining Software and Change Request Repositories". In: Proceedings of the 11th IEEE International Software Metrics, 2005, 29p.

[Cha00] Chapin, N. “Do we know what preventive maintenance is?”. In: Proceedings International Conference on Software Maintenance, IEEE Comput. Soc. Press, 2000, pp. 15-17.

[Che01] Chen, A.; Chou, E.; Wong, J.; Yao, A. Y.; Zhang, Q.; Zhang, S.; Michail, A. "CVSSearch : Searching through Source Code using CVS Comments". IEEE International Conference on Software Maintenance, 2001, pp. 364-379.

[Cim06] Cimiano, P. "Ontology learning and population from text: algorithms, evaluation and applications". Springer, 2006, 375p.

[Coc01] Cocchiarella, N. B. "Logic and ontology". Axiomathes, 2001, pp. 117-150.

[Cre94] Creswell, J. W. “Research design: Qualitative and quantitative approaches”. Thousand Oakes, Sage Publications, Inc, 1994, 248p.

[Cro10] Croft, W. B.; Metzler, D.; Strohman, T. "Search engines: information retrieval in practice". Addison-Wesley, 2010, 552 p.

[Dam12] DAML.org. "The DARPA Agent Markup Language Homepage". Capturado em: http://www.daml.org/, Maio 2012.

[Ecl12] Eclipse.org. “IDE Eclipse”. Capturado em: http://www.eclipse.org/, Junho 2012.

[Erl00] Erlikh, L. “Leveraging Legacy System Dollars for E-Business”. IT Professional, 2000, pp.17-23.

[Fra92] Frakes, W. B.; Baeza, Y. R. "Information Retrieval: Data Structures and Algorithms". Prentice Hall, 1992, 464p.

[Fre81] Freedman, D. P.; Weinberg, G.M. “A checklist for potential side effects of a maintenance change”. In Techniques of Program and System Maintenance, 1981, 56p.

[Gal91] Gallagher, K.B.; Lyle, J. R. “Using Program Slicing in Software Maintenance”. In IEEE Transactions on Software Engineering, 1991, pp. 751-761.

[Gil02] Gil, A.C. “Como elaborar projetos de pesquisa”. Atlas, 4a. Edição, 2002, 171p.

[Gom04] Gómez-Pérez, A.; Fernández-López, M.; Corcho, O. "Ontological engineering: with examples from the areas of knowledge management, e-commerce and the Semantic Web". Springer, 2004, 415p.

[Goo12] Google.com "Google". Capturado em: http://www.google.com/, Maio 2012.

[Got94] Gotel, O. C. Z.; Finkelstein, A. C. W. “Modelling the Contribution Structure Underlying Requirements Relation Between Requirements Traceability & Software”. In: First International Workshop on Requirements Engineering, 1994, pp. 71-81.

[Gru93] Gruber, T. R. “A translation approach to portable ontology specifications”. Knowl. Acquis., 1993, pp.199-220.

[Gru95] Gruber, T. R. “Toward principles for the design of ontologies used for knowledge sharing”. Int. J. Hum.-Comput. Stud., 1995, pp.907-928.

[Gse90] IEEE 610.12-1990 “IEEE Standard Glossary of Software Engineering Terminology”. Capturado em: http://standards.ieee.org/findstds/standard/610.12-1990.html, Junho 2012.

[Gua98] Guarino, N. “Formal ontology and information systems”. In Proceedings of FOIS’98, IOS Press. 1998, pp.3-15.

[Han72] Haney, F. M. "Module connection analysis. A tool for scheduling software debugging activities". Fall Joint Computer Conference, 1972, pp. 173-179.

[Has02] Hass, A. M. J. “Configuration Management Principles and Practice”. Addison-Wesley Professional, 2002, 432p.

[Hav03] Haveliwala, T. H. "Topic-Sensitive PageRank : A Context-Sensitive Ranking Algorithm for Web Search". Knowledge Creation Diffusion Utilization, 2003, pp.784-796.

[Hel91] Helm, R.; Maarek, Y. S. "Integrating information retrieval and domain specific approaches for browsing and retrieval in object-oriented class libraries". SIGPLAN, 1991, pp. 47-61.

[Hol07] Holland, G.; Lycett, M. “Semantic-Based Systems Development”. Conference on Object Oriented Programming Systems Languages and Applications, 2007, pp. 593-603.

[Jav12] Java. “java.com”. Capturado em: http://www.java.com/, Junho 2012.

[Jdt12] JDT “Java Development Tools”. Capturado em: http://www.eclipse.org/jdt/overview.php, Junho 2012.

[Jen12] Jena “Jena – A Semantic Web Framework for Java”. Capturado em: http://jena.sourceforge.net, Junho 2012.

[Jon07] Jonsson, P. "Exploring Process Aspects of Change Impact Analysis". PhD. Thesis, Blekinge Institute of Technology, 2007, 270p.

[Jun12] JUNG “Java Universal Network/Graph Framework”. Capturado em: http://jung.sourceforge.net/, Junho 2012.

[Jfr12] JavaFree.org “Ex prático(Project in NetBeans): Conceito MVC+Servlet+JSP”. Capturado em: http://javafree.uol.com.br/artigo/869464/Ex-praticoProject-in-NetBeans-Conceito-MVC+Servlet+JSP.html.

[Jwn12] JWNL “JWNL: Java WordNet Library”. Capturado em: http://sourceforge.net/projects/jwordnet/, Junho 2012.

[Kan00] Kantrowitz, M.; Street, H. "Stemming and its effects on TFIDF Ranking" In: Proc of ACM SIGIR, 2000, pp.357 - 359.

[Kar99] Karp, P. D.; Chaudhri, V. K.; Thomere, J. F. "XOL: An XML-Based Ontology Exchange Language". Technical Report 559, AI Center, SRI International, 1999, 32p.

[Kas04] Kasse, T. “Practical Insight into Cmmi”, Artech House Publishers, 2004, 472p.

[Kif98] KIF. "Knowledge Interchange Format". Capturado em: http://logic.stanford.edu/kif/dpans.html, Maio 2012.

[Kit04] Kitchenham, B. “Procedures for performing systematic reviews”. Technical report,Keele University, 2004, 33p.

[Kit95] Kitchenham, B.; Pickard, L.; Pfleeger, S. L. “Case Studies for Method and Tool Evaluation”. IEEE Software, 1995, pp. 52-62.

[Kle99] Kleinberg, J. M. "Authoritative Sources in a Hyperlinked Environment". In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 604 - 632.

[Kor04] Korb, K.B.; Nicholson, A.E. "Bayesian Artificial Intelligence". Chapman and Hall, CRC, 2004, 392p.

[Kot98] Kotonya, G.; Sommerville, I. "Requirements engineering: processes and techniques". J. Wiley, 1998, 294p.

[Kru03] Kruchten, P. “The Rational Unified Process: An Introduction”. Addison-Wesley Longman Publishing, 2003, 320p.

[Kuh07] Kuhn, A.; Gı, T. "Semantic clustering : Identifying topics in source code". Information and Software Technology, 2007, pp. 230-243.

[Kum05] Kumar, R. “Research Methodology: A Step-by-Step Guide for Beginners”, Sage Publications Ltd, 2005, 352p.

[Lam98] Landauer, T. K.; Foltz, P. W.; Laham, D. "An introduction to latent semantic analysis". Discourse Processes, 1998, pp. 259-284.

[Law03] Law, J.; Rothermel, G. “Whole Program Path-Based Dynamic Impact Analysis”. In: Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 308-318.

[Lef99] Leffingwell, D.; Widrig, D. "Managing Software Requirements: A Unified Approach". Addison-Wesley, 1999, 528p.

[Leh96] Lehman, M. M., “Laws of Software Evolution Revisited”. In: Proc of the 5th European Workshop on Software Process Technology, Springer-Verlag, 1996, pp. 108-124.

[Lev66] Levenshtein, V., "Binary codes capable of correcting deletions, insertions, and reversals" Soviet Physics Doklady, 1966, pp. 707-710.

[Lin94] Lindvall, M. “A Study of Traceability in Object-Oriented Systems Development”. Linköping University, 1994, 150p.

[Lin98] Lindvall, M. "How well do experienced software developers predict software change?" Journal of Systems and Software, 1998, 19-27.

[Lov68] Lovins J. B, "Development of a Stemming Algorithm". Mechanical Translation and computation Linguistics, 1968, pp. 22-31.

[Luc07] Lucia, A.D.; Fasano, F.; Oliveto, R.; Tortora, G. “Recovering traceability links in software artifact management systems using information retrieval methods”. ACM Transactions on Software Engineering and Methodology, 2007, pp. 13-50.

[Luc11] Lucia, A. D.; Penta, M. D.; Oliveto, R. "Improving Source Code Lexicon via Traceability and Information Retrieval". IEEE Transactions on Software Engineering, 2011, pp. 205-227.

[Mad01] Madhavan, J.; Bernstein, P.A.; Rahm, E. "Generic Schema Matching with Cupid" In: Proceedings of the 27th International Conference on Very Large Data Bases, 2001, pp. 49-58.

[Mae02] Maedche, A. “Ontology Learning for the Semantic Web”. Willey, 2002, 272p.

[Mar01] Marcus, A.; Maletic, J. I. "Identification of High-Level Concept Clones in Source Code". In: Proceedings of the 16th IEEE international conference on Automated software engineering, 2001, pp. 107-115.

[Mar04a] Marcus, A. "Semantic driven program analysis". In: 20th IEEE International Conference on Software Maintenance, 2004, pp. 469-473.

[Mar04b] Marcus, A.; Sergeyev, A.; Rajlich, V.; Maletic, J. I. "An Information Retrieval Approach to Concept Location in Source Code". In: Proceedings of the 11th Working Conference on Reverse Engineering, (2004, pp. 214-223.

[Mar09] Marmanis, H.; Babenko, D. "Algorithms of the Intelligent Web". Manning Publications Co., 2009, 368p.

[Mau12] Maui-Indexer “Multi-purpose automatic topic indexing”. Capturado em: http://code.google.com/p/maui-indexer/, Junho 2012.

[Mem12] Memoranda. “Memoranda, personal diary and scheduling tool”. Capturado em: http://memoranda.sourceforge.net/, Junho 2012.

[Men12] Mendeley. “Mendeley - free reference manager and academic social network”. Capturado em: http://www.mendeley.com/, Junho 2012.

[Nar11] Narayan, N.; Bruegge, B.; Delater, A.; Paech, B. "Enhanced traceability in model-based CASE tools using ontologies and information retrieval". 4th International Workshop on Managing Requirements Knowledge, 2011, pp. 24-28.

[Nau69] Naur, P.; Randell, B. "Software Engineering". Report of a conference sponsored by the NATO Science Committee, 1969, 16p.

[Nea02] Neal, J. S. O.; Carver, D. L. “Analyzing the Impact of Changing Requirements. Software Maintenance”, In: Proceedings IEEE International Conference on, 2002, pp. 190-195.

[Nol07a] Noll, R. P. “Rastreabilidade Ontológica sobre o Processo Unificado”. Dissertação de mestrado, Faculdade de Informática - PUCRS, 2007, 116 p.

[Nol07b] Noll, R. P.; Blois, M. R. “Ontological Traceability over the Unified Process”. ECBS2007 - 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2007, pp. 249-255.

[Nol07c] Noll, R. P.; Blois, M. R. “Enhancing Traceability using Ontologies”. SAC2007 - The 22nd Annual ACM Symposium on Applied Computing, 2007, pp. 1496-1497.

[Nol07d] Noll, R. P.; Saccol, D. B.; Edelweiss, N. “Uma proposta para análise de similaridade entre documentos XML e ontologias em OWL”. In: Proceedings Simpósio Brasileiro de Banco de Dados, 2007, pp. 2005-2008.

[Nol08] Noll, R. P.; Blois, M. B. “Usando experimentos para avaliar o mapeamento automático entre conceitos do domínio e especificação de Software”. In: V Experimental Software Engineering Latin American Workshop, 2008, 7p.

[Nol09] Noll, R.P.; Blois, M.R. “Concepts-Based Traceability: Using Experiments to Evaluate Traceability Techniques”. In: International Conference on Enterprise Information (ICEIS), 2009, pp. 539-550.

[Nol11] Noll, R. P. “Um Modelo para a Análise de Impacto em Código Fonte usando Ontologias”. Proposta de Tese, Faculdade de Informática - PUCRS, 2011, 101p.

[Oat06] Oates, B. “Researching Information Systems and Computing”. Sage Publications Ltd, 2006, 341p.

[Odm12] ODM. “OMG Ontology Definition Metamodel”. Capturado em: http://www.omg.org/ontology/, Junho 2012.

[Oli08] Oliveto, R. "Traceability Management Meets Information Retrieval Methods: Strengths and Limitations". ReCALL. University of Salermo, 2008, pp. 302-305.

[Oli10] Oliveira, A. F. “Change impact analysis from business rules”. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 2010, pp. 353-354.

[Ols08] Olson, D.L.; Delen, D. “Advanced Data Mining Techniques”, Springer, 2008, 192p.

[Ors03] Orso, A.; Apiwattanapong, T.; Harrold, M. J. “Leveraging field data for impact analysis and regression testing”. ACM SIGSOFT Software Engineering Notes 28, 2003, pp. 128 - 137.

[Owg12] OWG. “Ontology Working Group”. Capturado em: http://ontology.omg.org/, Junho 2012.

[Pal00] Palmer, J.D. “Traceability”. In Software Requirements Engineering. IEEE Computer Society Press, 2000, pp. 412-422.

[Pet09] Petrenko, M.; Rajlich, V. “Variable Granularity for Improving Precision of Impact Analysis”, IEEE International Conference on Program Comprehension, 2009, pp. 10-19.

[Pfl04] Pfleeger, S. L., “Engenharia de Software: Teoria e Prática”. Prentice Hall, 2004, 560p.

[Pfl90] Pfleeger, S.L.; Bohner, S. A. “A framework for software maintenance metrics”. Conference on Software Maintenance, 1990, pp. 320-327.

[Piv05] Pivk, A.; Cimiano, P.; Sure, Y. "From Tables to Frames". Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 2005, pp. 132-146.

[Pmb08] PMI. “PMBoK: Um guia do conhecimento em gerenciamento de projetos". Project Management Institute, 2008, 492p.

[Poh96] Pohl, K. "Process-Centered Requirements Engineering". John Wiley Research Science Press, 1996, 360p.

[Pos06a] Poshyvanyk, D.; Petrenko, M.; Marcus, A.; Xie, X.; Liu, D. "Source Code Exploration with Google". In: 22nd IEEE International Conference on Software Maintenance, 2006, pp. 334-338.

[Pos06b] Poshyvanyk, D.; Gueheneuc, Y. G.; Marcus, A.; Antoniol, G.; Rajlich, V. "Combining probabilistic ranking and latent semantic indexing for feature identification". In: 14th IEEE International Conference on Program Comprehension, 2006, pp. 137-148.

[Pos07] Poshyvanyk, D.; Marcus, A. "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code". In: 15th IEEE International Conference on Program Comprehension, 2007, pp. 37-48.

[Rad86] Rome Air Development Center. “Automated Life Cycle Impact Analysis System”, Tech. Report RADC- TR-86-197, Rome Laboratories, Rome, N.Y., 1986, 46p.

[Ram01] Ramesh, B.; Jarke, M. "Toward Reference Models for Requirements Traceability". IEEE Transactions on Software Engineering, 2001, pp. 58-93.

[Rdf04] RDF. “Resource Description Framework”. Capturado em: http://www.w3.org/RDF/, Junho 2012.

[Ril06] Rilling, J.; Zhang, Y.; Meng, W. J.; Witte, R. "A Unified Ontology-Based Process Model for Software Maintenance and Comprehension". In: Proceedings of the international conference on Models in software engineering, 2006, pp. 1-10.

[Roy70] Royce, W. W. "Managing the development of large software systems: concepts and techniques". Electronics, 1970, pp.1-9.

[Rug90] Rugaber, S.; Ornburn, S.B.; LeBlanc J. “Recognizing Design Decisions in Programs”. IEEE Softw., 1990, pp.46-54.

[Sal87] Salton, G.; Buckley, C. "Term Weighting Approaches in Automatic Text Retrieval". Information Processing and Management, 1987, pp. 513-523.

[She07] Shepherd, D.; Fry, Z. P.; Hill, E.; Pollock, L.; Vijay-Shanker, K. "Using natural language program analysis to locate and understand action-oriented concerns.: In: Proceedings of the 6th international conference on Aspect-oriented software development, 2007, pp. 212-224.

[Sho12] SHOE. “Simple HTML Ontology Extensions”. Capturado em: http://www.cs.umd.edu/projects/plus/SHOE/, Junho 2012.

[Sin06] Sindhgatta, R. "Using an information retrieval system to retrieve source code samples". In: Proceeding of the 28th international conference on Software engineering, 2006, pp. 905-908.

[Sjo02] Sjoberg D.; Anda B.; Arisholm, E.; Dyba, T.; Jorgensen, M.; Karahasanovic, A.; Koren, E.; Vokac, M. "Conducting Realistic Experiments in Software Engineering". In: Proceedings ISESE’2002, 2002, pp. 17-26.

[Som09] Sommerville, I., 2009. “Engenharia de Software”. Addison Wesley, 592p.

[Son08] Song, H. H.; Zhang, Z. X. “Study on Approach of Software Maintenance Based on Ontology Evolution”. International Conference on Computer Science and Software Engineering, 2008, pp. 585-588.

[Sps12] SPSS. “SPSS Statistical Analysis”. Capturado em: http://www.spss.com.br, Junho 2012.

[Sri92] Srinivasan, P. "Thesaurus construction". Prentice-Hall, 1992, pp. 161-218.

[Ste79] Stevens, W.; Myers, G.; Constantine, L. “Structured design”. In Classics in software engineering, Yourdon Press, 1979, pp. 205-232.

[Tan11] Tang, A.; Vliet, H. V. "Software Architecture Documentation : The Road Ahead". IEEE/IFIP Conference on Software Architecture, 2011, pp. 252-255.

[Tip95] Tip, F. “A Survey of Program Slicing Techniques”. Journal of Programming Languages, 1995, pp.1-65.

[Tra02] Travassos, G.H.; Gurov, D.; Amaral, G. "Introdução a Engenharia de Software Experimental". Relatório Técnico, PESC 590/02 – COPE/UFRJ, 2002, 52p.

[Vol01] Volz, R.; Oberle, D.; Staab, S.; Studer, R. "OntoLiFT Prototype". Communities, 2001, 28p.

[W3c12] W3C. “The World Wide Web Consortium”. Capturado em: http://www.w3.org, Junho 2012.

[Wei79] Weiser, M. D. "Program slices: formal, psychological, and practical investigations of an automatic program abstraction method". PhD thesis, University of Michigan, Ann Arbor, 1979, 215p.

[Wei81] Weiser, M. D. “Program Slicing”. In: ICSE ’81 Proceedings of the 5th international conference on Software engineering, 1981, pp. 439-449.

[Wel97] Welty, C. A. “Augmenting abstract syntax trees for program understanding”. In: Proceedings 12th IEEE International Conference Automated Software Engineering, 1997, pp. 126-133.

[Whi04] Whitman, M. E.; Woszczynski, A. B. "The handbook of information systems research”, Idea Group Pub, 2004, 364p.

[Woh00] Wohlin, C.; Runeson, P.; Host, M.; Ohlsson, M.; Regnell, B.; Wesslén, A. "Experimentation in software engineering: an introduction". Kluwer Academic Publishers, USA, 2000, 228p.

[Wor12] WordNet. “WordNet - A lexical database for the English language”. Capturado em: http://wordnet.princeton.edu/, Junho 2012.

[Yau78] Yau, S. S.; Colofello, J.S.; MacGregor, T. "Ripple effect analysis of software maintenance". In: I. C. S. Press Computer Software and Applications Conf. COMPSAC, 1978, pp. 60-65.

[Yau80] Yau, S.S.; Collofello, J.S. “Some Stability Measures for Software Maintenance”. In: IEEE Transactions on Software Engineering, 1980, pp. 545-552.

[Yin03] Yin, R.K. "Applications of case study research". Sage Publications, 2003, 200p.

[Zha06] Zhao, W.; Zhang, L.; Liu, Y.; Sun, J.; Yang, F. "SNIAFL: Towards a static noninteractive approach to feature location". ACM Trans. Softw. Eng. Methodol., 2006, pp. 195-226.

[Zha08a] Zhang, Y.; Witte, R.; Rilling, J.; Haarslev, V. “Ontological approach for the semantic recovery of traceability links between software artefacts”. Special issue on Language Engineering, 2008, pp. 185-203.

[Zha08b] Zhai, C. "Statistical Language Models for Information Retrieval". Morgan and Claypool Publishers, 2008, 146p.

APÊNDICE A. ARTIGOS DA REVISÃO SISTEMÁTICA.

Ref.	Eliminado no Estágio	Classif.	Referência
SEL01	Selecionado	C1	Aryani, A., Perin, F., Lungu, M., Mahmood, A. N., & Nierstrasz, O. (2011). Can we predict dependencies using domain information ? 18th Working Conference on Reverse Engineering. doi:10.1109/WCRE.2011.17
SEL02	Selecionado	C1	Dit, B., Revelle, M., Gethers, M., & Poshyvanyk, D. (2011). Feature location in source code: a taxonomy and survey. JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE. doi:10.1002/smr
SEL03	Selecionado	C1	Narayan, N., Bruegge, B., Delater, A., & Paech, B. (2011). Enhanced traceability in model-based CASE tools using ontologies and information retrieval. 2011 4th International Workshop on Managing Requirements Knowledge (pp. 24-28). Ieee. doi:10.1109/MARK.2011.6046559
SEL04	Selecionado	C1	Rilling, J., Zhang, Y., Meng, W. J., & Witte, R. (n.d.). A Unified Ontology-Based Process Model for Software Maintenance and Comprehension. Viewpoints (pp. 1-10).
SEL05	Selecionado	C2	Borg, M. (2011). Do Better IR Tools Improve the Accuracy of Engineers ' Traceability Recovery ? MALETS '11 (pp. 27-34).
SEL06	Selecionado	C2	Kuhn, A., & Gı, T. (2007). Semantic clustering : Identifying topics in source code. Information and Software Technology, 49, 230-243. doi:10.1016/j.infsof.2006.10.017
SEL07	Selecionado	C2	Lucia, A. D., Fasano, F., & Oliveto, R. (2008). Traceability Management for Impact Analysis. FoSM 2008 (pp. 21-30).
SEL08	Selecionado	C2	Tang, A., & Vliet, H. V. (2011). Software Architecture Documentation : The Road Ahead. IEEE/IFIP Conference on Software Architecture (pp. 252-255). doi:10.1109/WICSA.2011.40
EXCL01	5	C2	Lucia, A. D., Fasano, F., & Oliveto, R. (2008). **Traceability Management for Impact Analysis**. FoSM 2008 (pp. 21-30).
EXCL02	5	C1	Dit, B., Revelle, M., Gethers, M., & Poshyvanyk, D. (2011). **Feature location in source code: a taxonomy and survey**. JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE. doi:10.1002/smr
EXCL03	5	C2	Asuncion, H. U., & Taylor, R. N. (2009). Capturing Custom Link Semantics among Heterogeneous Artifacts and Tools. TEFSE'09 (pp. 1-5).
EXCL04	5	C2	Würsch, M., Ghezzi, G., Reif, G., & Gall, H. C. (n.d.). Supporting Developers with Natural Language Queries Categories and Subject Descriptors. ICSE '10 (pp. 165-174).
EXCL05	5	C1	Aversano, L., Marulli, F., & Tortorella, M. (2010). Recovering Traceability Links between Business Process and Software System Components. 18th IEEE International Conference on Program Comprehension (pp. 52-53). doi:10.1109/ICPC.2010.54
EXCL06	4	C2	Berg, K. V. D., & Hernández, J. (2006). Analysis of Crosscutting across Software Development Phases based on Traceability. Engineering.
EXCL07	4	C2	Chen, X. (2010). Extraction and Visualization of Traceability Relationships between Documents and Source Code. ASE'10 (pp. 505-509).
EXCL08	4	C2	Lima, E. J. C. D., Nt, J. A. R., Xexéo, G. B., & Souza, J. M. D. (2010). ARARA – A Collaborative Tool to Requirement Change Awarenes. Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design (pp. 134-139).
EXCL09	4	C2	Petrenko, M., Rajlich, V., & Vanciu, R. (2008). Partial Domain Comprehension in Software Evolution and Maintenance. Science, 13-22. doi:10.1109/ICPC.2008.14
EXCL10	4	C2	Taduri, S., Lau, G. T., Law, K. H., & Kesan, J. P. (2011). Retrieval of Patent Documents from Heterogeneous Sources using Ontologies and Similarity Analysis. IEEE International Conference on Semantic Computing (pp. 538-545). doi:10.1109/ICSC.2011.34
EXCL11	3	C4	Aleman-Meza, B., Nagarajan, M., Ding, L., Sheth, A., Arpinar, I. B., Joshi, A., & Finin, T. (2008). Scalable semantic analytics on social networks for addressing the problem of conflict of interest detection. ACM Transactions on the Web, 2(1), 1-29. doi:10.1145/1326561.1326568
EXCL12	3	C4	Antal, P., & Meszaros, T. (2001). Annotated Bayesian Networks: a Tool to Integrate Textual and Probabilistic Medical Knowledge. Ultrasound in

			Obstetrics and Gynecology, 177-182.
EXCL13	3	C4	Apostolou, D., Mentzas, G., Stojanovic, L., Thoenssen, B., & Pariente, T. (2011). A collaborative decision framework for managing changes in e-Government services. Government Information Quarterly, 28(1), 101-116. Elsevier Inc. doi:10.1016/j.giq.2010.03.007
EXCL14	3	C4	Batouche, B. (2011). Web Service Complex Request Ontology and its Answers Models. Constraints, 55-60.
EXCL15	3	C4	Belis, T. (2009). Distributed Multimedia Metadata Tracking and Management, 1-4.
EXCL16	3	C4	Boer, R. C. D., & Vliet, H. V. (2008). Architectural knowledge discovery with latent semantic analysis : Constructing a reading guide for software product audits. Architecture, 81, 1456-1469. doi:10.1016/j.jss.2007.12.815
EXCL17	3	C4	Boran, A., Sullivan, D. O., & Wade, V. (2010). A dependency modeling approach for the management of ontology based integration systems. Data Engineering, 914-917.
EXCL18	3	C4	Boustil, A., Sabouret, N., & Paris, L. I. D. (2010). Web Services Composition Handling User Constraints : Towards a Semantic Approach. Online, 8-10.
EXCL19	3	C4	Braga, R. M. M., Werner, M. L., & Mattoso, M. (2006). Odyssey-Search : A multi-agent system for component information search and retrieval. System, 79, 204-215. doi:10.1016/j.jss.2005.05.002
EXCL20	3	C4	Brien, W. O. (2013). The Journal of Systems and Software Avoiding semantic and temporal gaps in developing software intensive systems. Journal of Systems and Software, 81(2008), 1997-2013. doi:10.1016/j.jss.2008.01.041
EXCL21	3	C4	Canfora, G., & Penta, M. D. (2007). New Frontiers of Reverse Engineering New Frontiers of Reverse Engineering. Reverse Engineering.
EXCL22	3	C4	Carlsson, C., & Fedrizzi, M. (2011). A Hierarchical Approach to Assess Keyword Dependencies in Fuzzy Keyword Ontologies. October, 1799-1804.
EXCL23	3	C4	Chen, C., Song, I.-yeol, Yuan, X., & Zhang, J. (2008). The thematic and citation landscape of Data and Knowledge Engineering (1985 – 2007) q. Data & Knowledge Engineering, 67, 234-259. doi:10.1016/j.datak.2008.05.004
EXCL24	3	C4	Cheng, B. H. C., Atlee, J. M., & Joanne, M. (2009). Research Directions in Requirements Engineering Research Directions in Requirements Engineering. Requirements Engineering.
EXCL25	3	C4	Chua, C. E. H., Purao, S., & Storey, V. C. (2006). Developing maintainable software : The Readable approach. Decision Support Systems, 42, 469-491. doi:10.1016/j.dss.2005.04.002
EXCL26	3	C4	Crasso, M., Zunino, A., & Campo, M. (n.d.). Query by Example for Web Services. Text, 2376-2380.
EXCL27	3	C4	Dibley, M. J., Li, H., Miles, J. C., & Rezgui, Y. (2011). Advanced Engineering Informatics Towards intelligent agent based software for building related decision support. Advanced Engineering Informatics, 25, 311-329. doi:10.1016/j.aei.2010.11.002
EXCL28	3	C4	Dong, A., & Agogino, A. M. (1996). AID 96 Prize Paper Text analysis for constructing design representations *. Intelligence.
EXCL29	3	C4	Gui, Z., Wu, H., & Wang, Z. (2008). A Data Dependency Relationship Directed Graph and Block Structures Based Abstract Geospatial Information Service Chain Model, 21-27. doi:10.1109/NCM.2008.183
EXCL30	3	C4	Haiduc, S., & Marcus, A. (2010). Supporting Program Comprehension with Source Code Summarization. Source, 223-226.
EXCL31	3	C4	Hindle, A., Godfrey, M. W., & Ernst, N. A. (2011). Automated Topic Naming to Support Cross-project Analysis of Software Maintenance Activities. Design, 163-172.
EXCL32	3	C4	Ienco, D., Pensa, R. G., & Meo, R. (2012). From Context to Distance: Learning Dissimilarity for Categorical Data Clustering. ACM Transactions on Knowledge Discovery from Data, 6(1), 1-25. doi:10.1145/2133360.2133361
EXCL33	3	C4	Jalali, V., Reza, M., & Borujerdi, M. (2011). Information retrieval with concept-based pseudo-relevance feedback in MEDLINE. Knowledge and Information Systems, 237-248. doi:10.1007/s10115-010-0327-7
EXCL34	3	C4	Janev, V., & Vraneš, S. (2011). Applicability assessment of Semantic Web technologies. Information Processing and Management, 47(4), 507-517. Elsevier Ltd. doi:10.1016/j.ipm.2010.11.002
EXCL35	3	C4	Janssens, F., Leta, J., Gla, W., & Moor, B. D. (2006). Towards mapping library and information science. Journal of the American Society for Information Science, 42, 1614-1642. doi:10.1016/j.ipm.2006.03.025
EXCL36	3	C4	Kawaguchi, S. (2006). MUDABlue : An automatic categorization system for Open Source repositories. Journal of Systems and Software, 79, 939-953.

			doi:10.1016/j.jss.2005.06.044
EXCL37	3	C4	Khan, A., Baharudin, B., & Khan, K. (2010). Efficient Feature Selection and Domain Relevance Term Weighting Method for Document Classification. 2010 Second International Conference on Computer Engineering and Applications, 398-403. Ieee. doi:10.1109/ICCEA.2010.228
EXCL38	3	C4	Kiryakov, A., Popov, B., Terziev, I., Manov, D., & Ognyanoff, D. (2004). Semantic annotation , indexing , and retrieval. World Wide Web Internet And Web Information Systems, 2, 49-79. doi:10.1016/j.websem.2004.07.005
EXCL39	3	C4	Kohlhase, M., Corneli, J., David, C., Ginev, D., Jucovschi, C., & Kohlhase, A. (2011). The Planetary System : Web 3 . 0 & Active Documents for STEM. Computer, 4, 598-607. doi:10.1016/j.procs.2011.04.063
EXCL40	3	C4	Koike, A., & Takagi, T. (2005). PRIME : Automatically Extracted PR otein I nteractions and M olecular Information Databas E. In Silico Biology, 5(December 2004), 9-20.
EXCL41	3	C4	Larsen, K. R., Monarchi, D. E., Hovorka, D. S., & Bailey, C. N. (2008). Analyzing unstructured text data : Using latent categorization to identify intellectual communities in information systems. Structure, 45, 884-896. doi:10.1016/j.dss.2008.02.009
EXCL42	3	C4	Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. ACM Transactions on Information Systems, 25(2), 6-es. doi:10.1145/1229179.1229180
EXCL43	3	C4	Mchenry, W. K. (2002). Using knowledge management to reform the Russian Criminal Procedural Codex. Decision Support Systems, 34, 339-357.
EXCL44	3	C4	Mencía, E. L. (2009). Segmentation of Legal Documents. Seminar, 88-97.
EXCL45	3	C4	Mügge, H., & Cremers, A. B. (n.d.). Integrating Aspect-Orientation and Structural Annotations to Support Adaptive Middleware. Provider, 9-14.
EXCL46	3	C4	Orsi, G., Milano, P., Leonardo, P., Tanca, L., Zimeo, E., & Sannio, U. (n.d.). Keyword-based , Context-aware Selection of Natural Language Query Patterns Categories and Subject Descriptors. Search, (i), 189-200.
EXCL47	3	C4	Overbeek, S. J., Bommel, P. V., & Proper, H. A. E. (2011). Statics and dynamics of cognitive and qualitative matchmaking in task fulfillment. Information Sciences, 181(1), 129-149. Elsevier Inc. doi:10.1016/j.ins.2010.09.002
EXCL48	3	C4	Port, D., Nikora, A., & Hihn, J. (2011). Experiences with Text Mining Large Collections of Unstructured Systems Development Artifacts at JPL. Text, 701-710.
EXCL49	3	C4	Pérez-castillo, R., Guzmán, I. G.-rodríguez D., & Piattini, M. (2011). Computer Standards & Interfaces Knowledge Discovery Metamodel-ISO / IEC 19506 : A standard to modernize legacy systems. Computer Standards & Interfaces, 33(6), 519-532. Elsevier B.V. doi:10.1016/j.csi.2011.02.007
EXCL50	3	C4	Qasemizadeh, B., Shen, J., Neill, I. O., Miller, P., Hanna, P., Stwart, D., & Wang, H. (2009). A Speech Based Approach to Surveillance Video Retrieval. Analysis, 336-339. doi:10.1109/AVSS.2009.54
EXCL51	3	C4	Ramachandran, V., Gupta, M., Sethi, M., & Chowdhury, S. R. (2009). Determining Configuration Parameter Dependencies via Analysis of Configuration Data from Multi-tiered Enterprise Applications. Solutions, 169-178.
EXCL52	3	C4	Ruiz-garcia, L., Steinberger, G., & Rothmund, M. (2010). A model and prototype implementation for tracking and tracing agricultural batch products along the food chain. Food Control, 21(2), 112-121. Elsevier Ltd. doi:10.1016/j.foodcont.2008.12.003
EXCL53	3	C4	Schugerl, P. (2011). Scalable Clone Detection Using Description Logic. Change, 47-53.
EXCL54	3	C4	Schugerl, P., Rilling, J., Charland, P., Defence, R., & Valcartier, D. C. (2008). Mining Bug Repositories – A Quality Assessment. Artificial Intelligence, 1105-1110. doi:10.1109/CIMCA.2008.63
EXCL55	3	C4	Selvaretnam, B., & Belkhatir, M. (2011). Natural language technology and query expansion : issues , state-of-the-art and perspectives. Journal of Intelligent Information Systems. doi:10.1007/s10844-011-0174-3
EXCL56	3	C4	She, S., Lotufo, R., & Berger, T. (2011). Reverse Engineering Feature Models. Power, 461-470.
EXCL57	3	C4	Shirogane, J. (n.d.). Correspondence Validation Method for GUI Operations and Scenarios by Operation History Analysis. Interfaces, 257-266.
EXCL58	3	C4	Spek, P. V. D., & Klusener, S. (2011). Applying a dynamic threshold to improve cluster detection of LSI. Science of Computer Programming, 76(12), 1261-1274. Elsevier B.V. doi:10.1016/j.scico.2010.12.004

EXCL59	3	C4	Stumme, G. (2002). Computing iceberg concept lattices with Titanic. Data & Knowledge Engineering, 42, 189-222.
EXCL60	3	C4	Sánchez, D., Valls, A., & Gibert, K. (2010). Ontology-driven web-based semantic similarity. Journal of Intelligent Information Systems (pp. 383-413). doi:10.1007/s10844-009-0103-x
EXCL61	3	C4	Theobald, M., Bast, H., Majumdar, D., Schenkel, R., & Weikum, G. (2007). TopX: efficient and versatile top-k query processing for semistructured data. The VLDB Journal, 17(1), 81-115. doi:10.1007/s00778-007-0072-z
EXCL62	3	C4	Verginadis, Y., Papageorgiou, N., Apostolou, D., & Mentzas, G. (n.d.). A Review of Patterns in Collaborative Work. Management, 283-292.
EXCL63	3	C4	Wagner, T., Visser, U., & Herzog, O. (2004). Egocentric qualitative spatial knowledge representation for physical robots. Robotics and Autonomous Systems, 49, 25-42. doi:10.1016/j.robot.2004.07.022
EXCL64	3	C4	Wang, Y. (2008). Ontology Evolution Issues in Adaptable Information Management Systems. Scenario, 753-758. doi:10.1109/ICEBE.2008.69
EXCL65	3	C4	Yang, J., Zhang, S., & Jin, W. (n.d.). DELTA : Indexing and Querying Multi-labeled Graphs. Database, 1765-1774.
EXCL66	3	C4	Zhu, S., Wu, J., Xiong, H., & Xia, G. (2011). Scaling up top-K cosine similarity search. DATAK, 70(1), 60-83. Elsevier B.V. doi:10.1016/j.datak.2010.08.004
EXCL67	3	C4	Ã, B. A. G., Fredriksen, R., & Thunem, A. P. (2007). Addressing dependability by applying an approach for model-based risk assessment. Reliability Engineering, 92, 1492-1502. doi:10.1016/j.ress.2006.10.002
EXCL68	3	C3	Boran, A., Sullivan, D. O., & Wade, V. (2010). Managing Ontology Based Integration Systems using Dependencies. Transform, 165-170.
EXCL69	3	C3	Cleland-huang, J. (2005). Toward Improved Traceability of Non-Functional Requirements. Information Retrieval, 14-19.
EXCL70	3	C3	Dekhtyar, A., Hayes, J. H., & Larsen, J. (2007). Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools? Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), 4-4. Ieee. doi:10.1109/PROMISE.2007.8
EXCL71	3	C3	Jansen, A., Avgeriou, P., & Ven, J. S. V. D. (2009). The Journal of Systems and Software Enriching software architecture documentation. The Journal of Systems & Software, 82(8), 1232-1248. Elsevier Inc. doi:10.1016/j.jss.2009.04.052
EXCL72	3	C3	Medina-dominguez, F., Amescua, A. D., & Mora-soto, A. (2010). Improving the efficiency of use of software engineering practices using product patterns. Information Sciences, 180(14), 2721-2742. Elsevier Inc. doi:10.1016/j.ins.2010.03.028
EXCL73	3	C3	Niu, N., Ms, C., Easterbrook, S., Engineering, D. S., & Specifications, R. (n.d.). Concept Analysis for Product Line Requirements. Quality, 137-148.
EXCL74	3	C3	Okubo, T. (2011). Effective Security Impact Analysis with Patterns. doi:10.1109/ARES.2011.79
EXCL75	3	C3	Oliveto, R., Antoniol, G., Marcus, A., & Hayes, J. (2007). Software Artefact Traceability : the Never-Ending Challenge. Change, 485-488.
EXCL76	3	C3	Reiss, S. P. (2009). Semantics-Based Code Search. Science, 243-253.
EXCL77	3	C3	Schwarz, H. (2009). Towards a Comprehensive Traceability Approach in the Context of Software Maintenance, 339-342. doi:10.1109/CSMR.2009.8
EXCL78	3	C3	Tichy, W. F., Körner, S. J., & Landhäußer, M. (2010). Creating Software Models with Semantic Annotation. International Journal, 17-18.
EXCL79	3	C3	Vaduva, A., & Zijcker, R. (n.d.). A # - A Metamodel for Data Preprocessing. System.
EXCL80	3	C3	Yang, Y., & Lu, Q. (n.d.). A Clustering Based Approach for Domain Relevant Relation Extraction. Nature.
EXCL81	3	C3	Zhuge, H. (2004). Fuzzy resource space model and platform. Journal of Systems and Software, 73, 389-396. doi:10.1016/S0164-1212(03)00235-8

APÊNDICE B. ANÁLISE DE IMPACTO COM JRIPPLES

Para ilustrar o uso da ferramenta, a Figura A2.1 apresenta o início da análise na qual o usuário deve informar o projeto que será inspecionado, bem como a classe que inicia o sistema, isto é, aquela que possui o método estático *main*.

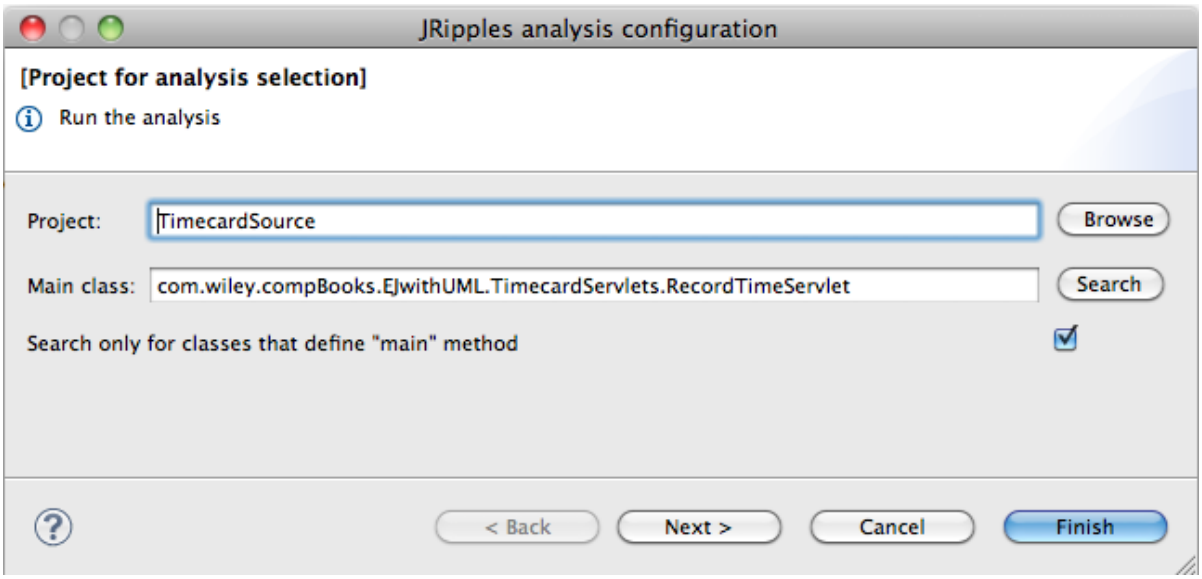


Figura 7.1 – Seleção inicial para análise de dependência usando JRipples.

Uma vez selecionado o projeto e a classe principal, a ferramenta apresenta uma lista de dependências, conforme Figura A2.2. O usuário deve abrir a classe marcada como *Next* e verificar se a mesma será impactada. Caso positivo, ele a marca como *Propagating* e avalia todas as classes marcadas como *Next*. Caso a classe não seja alterada, ele a marca como *Unchanged*. Caso ele encontre as estruturas que necessitam alguma alteração, a classe deve ser marcada como *Located*.

Conforme apresentado, a análise de dependência fornece os indícios de quais classes podem ser afetadas, baseada na inspeção de código e interpretação do desenvolvedor. O papel do programador é fundamental neste tipo de análise, pois ele deve selecionar, visitar, compreender e/ou modificar componentes associados a alguma requisição de mudança.

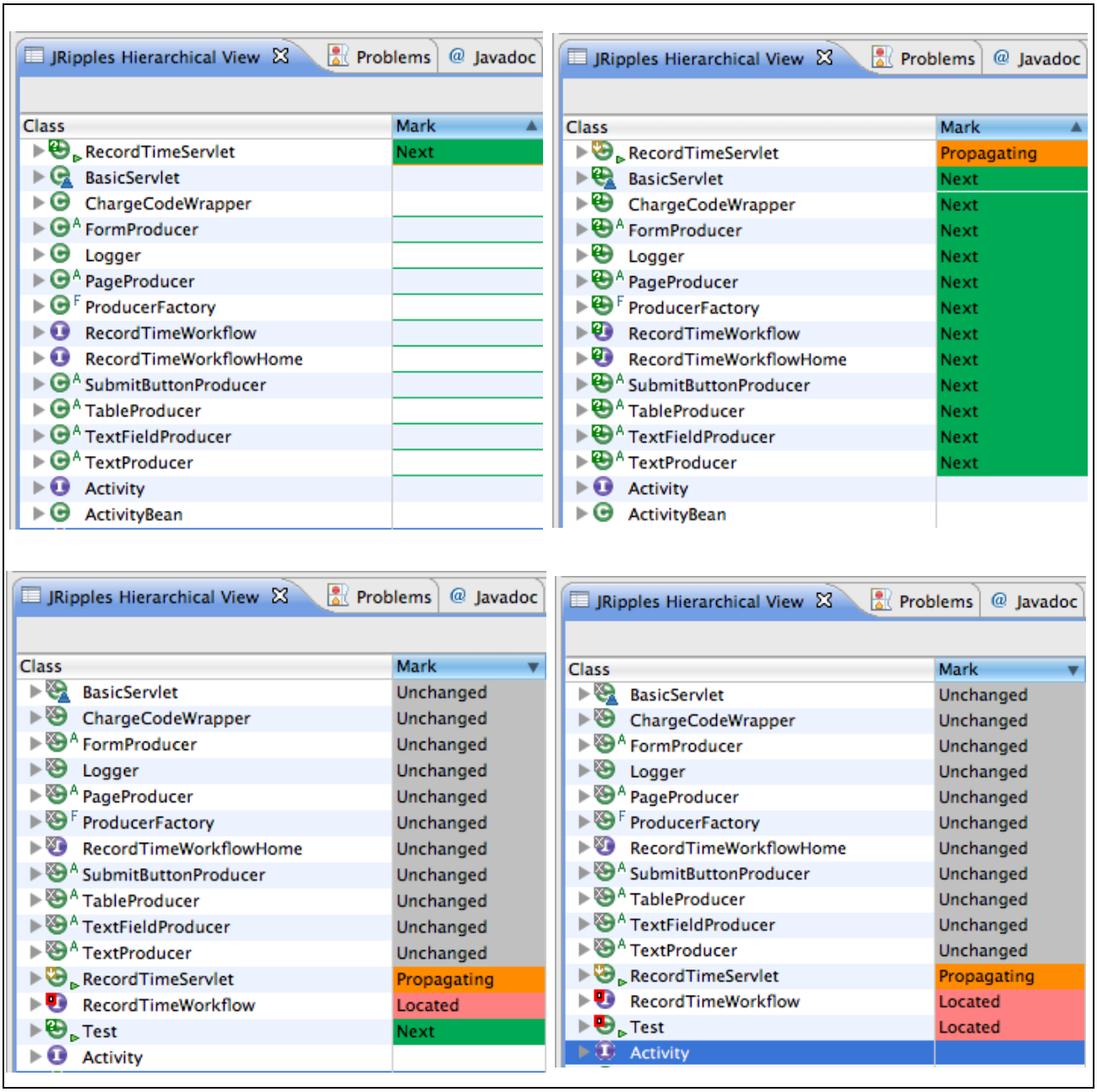


Figura 7.2 – Análise de dependência manual usando JRipples.

APÊNDICE C. INTEGRAÇÃO DE ONTOLOGIAS COM O DESENVOLVIMENTO DE SOFTWARE

A construção de uma ontologia não é uma atividade trivial e requer um conhecimento substancial sobre o domínio modelado (a partir de especialistas) e dos construtores de uma linguagem formal (a partir dos engenheiros de conhecimento). Com o objetivo de integrar as atividades relacionadas a manutenção de ontologias durante o desenvolvimento de software, é necessário adaptar esse processo para incluir atividades de engenharia, aprendizagem e evolução de ontologias.

A literatura fornece diferentes abordagens para o desenvolvimento de ontologias [Gruninger et al. 1995; Fernandez et al. 1997; Noy et al. 2001; Sure et al. 2002]. Analisando estas abordagens, as mesmas são baseadas em um processo iterativo que inclui as seguintes principais atividades: (1) definição do escopo da ontologia; (2) aquisição dos conceitos do domínio (modelagem conceitual); (3) formalização dos conceitos (modelagem formal); (4) reuso e integração; (5) definição dos axiomas; e (6) validação da ontologia.

Conforme [Maedche et al. 2002], apesar da maturidade dessas metodologias, o processo de aquisição de conhecimento continua uma atividade tediosa e manual, resultando em um dos gargalos para sua aquisição. É necessário um processo que integre boas práticas da engenharia ontológica [Gomez-Perez et al. 2004] e métodos de aprendizagem [Cimiano 2006] enquanto adapte seu escopo para capturar o conhecimento sobre os produtos de trabalhos do desenvolvimento de software. O objetivo do processo proposto é modelar o conhecimento e manter o relacionamento semântico entre os produtos de trabalho e o conhecimento do domínio. Para tanto o processo foi organizado em três grandes fases: Projeto, Manutenção e Verificação conforme apresentado na Figura A3.1.

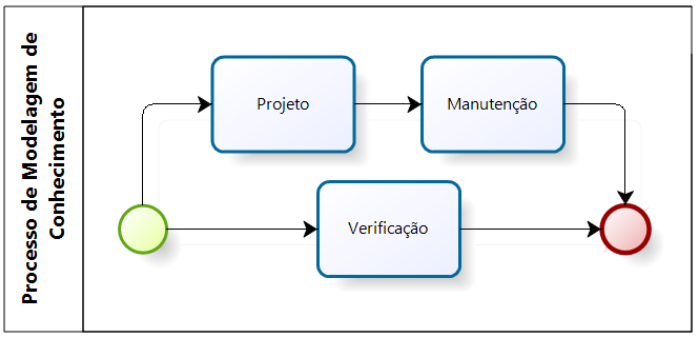


Figura 8.1 – *Workflow* de Processo de Modelagem de Conhecimento.

A modelagem de conhecimento inicia pela fase de Projeto, a qual define o escopo da ontologia, seus conceitos e propriedades taxonômicas (subClass e subProperty) e não taxonômicas (object e datatype properties). A Figura A3.2 apresenta o modelo de processo relacionado à fase de Projeto. Um novo papel chamado Engenheiro de Conhecimento (EC) é introduzido como o responsável primário pelas atividades de modelagem do conhecimento. As atividades do EC incluem refinar o Modelo de Domínio baseado nas propriedades taxonômicas e não taxonômicas. Estas atividades podem ser feitas utilizando uma ferramenta de edição de ontologias, baseado na ontologia gerada automaticamente a partir do Modelo de Domínio. Esta versão é automaticamente extraída a partir do Modelo de Domínio usando o mapeamento UML para OWL proposto pela OMG (*Object Management Group*) [ODM 2011] e sumarizado na Tabela A3.1. Esta tese utiliza exclusivamente OWL como linguagem de representação de ontologia devido sua recomendação oficial pela [W3C 2011].

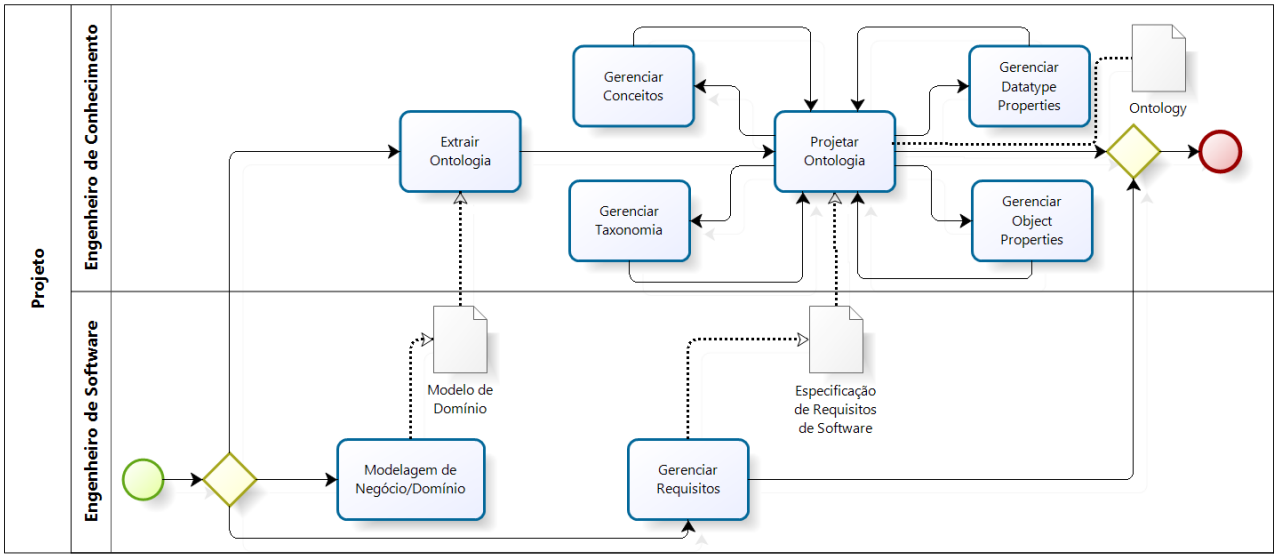


Figura 8.2 – *Workflow* de Projeto.

Tabela 8.1 – Mapeamento UML para OWL [ODM 2011]

UML	**OWL**
Classe	Classe OWL
Atributo	Datatype Property - Domínio: classe pai - Imagem: XSD definido pelo XML Schema
Associações	Object Property - Domínio: Classe de origem da associação - Imagem: Classe de destino da associação Restrições: 1. Se a associação inicial e final forem navegáveis, então a

	propriedade é simétrica 2. Se a multiplicidade de ambas as associações forem as mesmas, então gera uma restrição de cardinalidade. Caso contrário, gera uma restrição de cardinalidade mínima e máxima.
Generalização	Define a taxonomia entre as classes e propriedades OWL

Como o Modelo de Domínio representa um diagrama de classe conceitual, a Tabela A3.1 não inclui estereótipos, operações, parâmetros, valores de retorno e outros, pois os mesmos não devem ser inclusos neste modelo. O processo de refinamento também utiliza as especificações de requisitos de software para enriquecer os conceitos da ontologia, focando na extração de novos substantivos que frequentemente aparecem como candidatos a conceitos. Após o refinamento, a última atividade gera um novo produto de trabalho chamado Ontologia, representando a primeira versão da ontologia que captura o conhecimento fornecido pela disciplina de requisitos.

Esta tese não sugere a criação automática da Ontologia a partir dos requisitos funcionais e não funcionais nem de diagramas da UML, mas extrair os principais conceitos do Modelo de Domínio e representar essa conceituação utilizando práticas conhecidas como as apresentadas em [Gruninger et al. 1995; Fernandez et al. 1997; Noy et al. 2001; Sure et al. 2002]. Após o projeto da primeira versão da Ontologia, é necessário manter a consistência semântica dessa ontologia com os demais produtos de trabalho do desenvolvimento de software. Novos relacionamentos entre conceitos podem surgir durante a evolução do software, justificando modificações na ontologia desenvolvida durante a fase de Projeto.

A manutenção da ontologia é baseada em uma abordagem proposta por [Sure et al. 2002]. Os autores propõem um processo de manutenção em duas fases: a primeira diz respeito à análise da organização taxonômica para manter o núcleo da ontologia coerente com o conhecimento do domínio modelado, enquanto a segunda fase foca na inspeção e refinamento das relações não taxonômicas. A Figura A3.3 apresenta o principal fluxo de trabalho da atividade de Manutenção. O EC é responsável pelas atividades de Manutenção, recebendo como entrada a Ontologia, Especificação de Requisitos de Software e o Modelo de Análise e Projeto. O EC irá manter a Ontologia durante todo o ciclo de vida do software, refinando conceitos, propriedades e regras de derivação, bem como manter e validar os elos de rastreabilidade entre conceitos e diferentes produtos de trabalho.

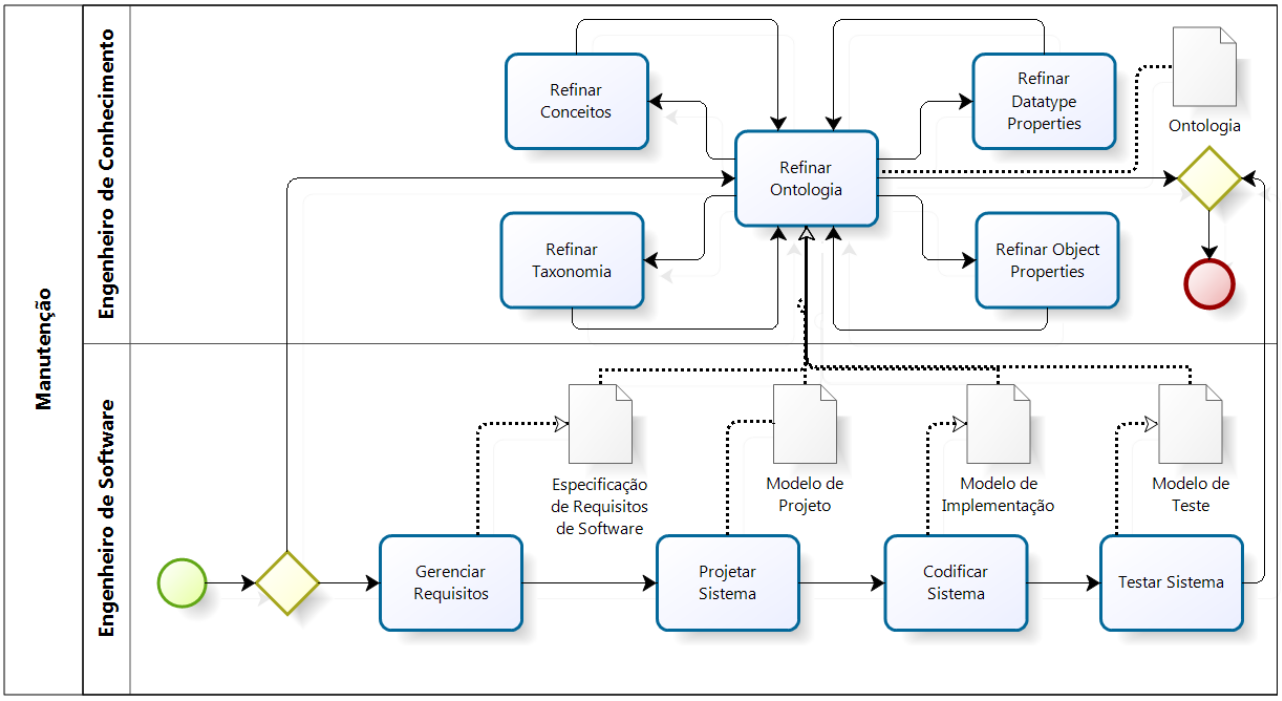


Figura 8.3 – *Workflow* de Manutenção.

O conhecimento da aplicação deve evoluir utilizando técnicas como Evolução de Ontologias [Stojanovic 2004], mantendo assim a exatidão das instâncias existentes dos elos de rastreabilidade durante qualquer manutenção do software e da ontologia. Cada modificação na Ontologia base pode gerar inconsistências lógicas com o conhecimento modelado. É necessário avaliar a integridade da ontologia em busca de inconsistências que podem invalidar a dedução lógica. A fase de verificação inclui as tarefas regulares que visam identificar estas inconsistências.

Adicionalmente, a consistência de modelos lógicos deve ser mantida utilizando uma abordagem indutiva e pragmática [Kishore et al. 2004] em cada iteração do ciclo de desenvolvimento. Isto significa que a ontologia é testada iterativamente para verificar se a mesma representa de forma adequada, coerente e consistente o domínio da aplicação. A proposta de [Gómez-Pérez 1996] pode ser utilizada como base para a fase de Verificação. O autor sugere que a arquitetura e definição de ontologias deve ser sólida, o léxico e a sintaxe devem ser corretos e o conteúdo das definições deve ser consistente, conciso, expansível e sensível. Isto significa que o EC deve identificar quais tipos de verificações devem ser executadas quando definições são adicionadas ou modificadas em uma ontologia. Como uma orientação comum, o EC deve verificar cada definição individual e axiomas, considerando aqueles explicitamente definidos e implicitamente inferidos a partir da ontologia.

A Figura A3.4 apresenta o fluxo de trabalho da Verificação. O EC recebe uma ontologia, o Modelo de Domínio, Especificação de Requisitos de Software e Modelo de Análise e Projeto com o objetivo de atualizar a ontologia, verificando sua consistência. Esta atividade inclui a verificação de conceitos, propriedades, regras semânticas de derivação, representando assim a evolução da ontologia baseada na evolução do software.

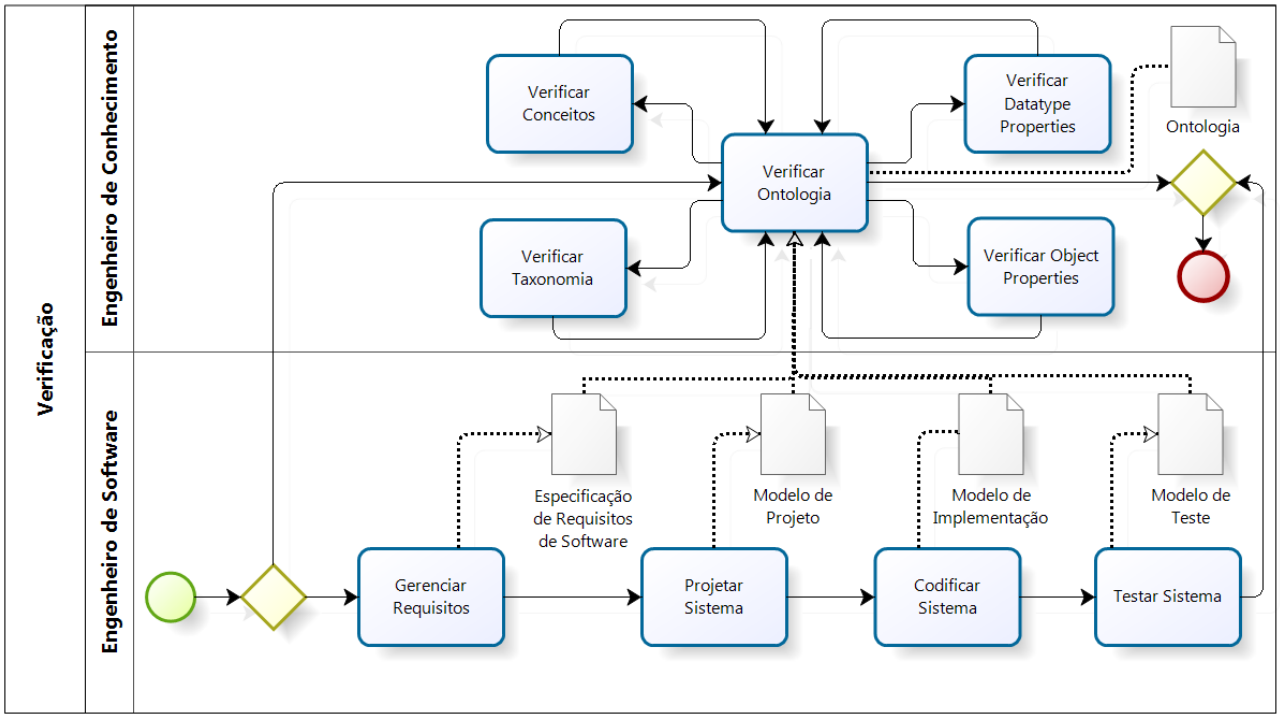


Figura 8.4 – *Workflow* de Verificação.

APÊNDICE D. ONTOLOGIA DE RASTREABILIDADE

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:j.0="http://iseg.pucrs.br/SEmantics#"
    xmlns="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#" >
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#worksIn">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextEntryFrame.TextEntryFrame"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TableProducer.getMaxColumnIndex"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TableProducer.getMaxRowIndex"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#getCurrentTimecard">
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.submitTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpServletRequest.getInputStream"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflow.submitTimecard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.getCurrentTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.setCurrentTimecard"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.TimecardBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.getCurrentTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.setCurrentTimecard"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardHome.findTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.addTimecard"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#name">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportFile.filename"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Node.name"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.name"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.projectName"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.clientName"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.name"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.chargeCodeName"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.name"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.name"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ComboBoxProducer.name"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Client"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextFieldProducer.name"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.filename"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.name"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#projectId">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.projectName"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.project"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#chargeCodeIds">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.chargeCodeName"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodes"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
```

```
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#register">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#hasActivity">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflow.getActivities"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.ActivityBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextEntryFrame.actionPerformed"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportFile.addEntry"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Test.loadActivityData"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextEntryFrame.TextEntryFrame"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowBean.getActivities"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#isFrom">
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LinkProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflowBean.isUserValid"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.isPasswordChangeRequired"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflowBean.isPasswordChangeRequired"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Client"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpServletRequest.isRequestedSessionIdFromUrl"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#SubmitButtonProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpServletRequest.isRequestedSessionIdFromCookie"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#PageProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#IConcreteProducer.distanceFromVersion"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TabularInputFormProducerGeneric.distanceFromVersion"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.isPasswordChangeRequired"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TableProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#FormProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflow.isUserValid"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextFieldProducerGeneric.distanceFromVersion"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.isPasswordValid"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.isPasswordValid"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LoginServlet.isLoginValid"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflow.isPasswordChangeRequired"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpSession.isNew"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpServletRequest.isRequestedSessionIdValid"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#newUser">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.employee"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.newUser"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.users"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#hasCurrent">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.setCurrentTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.getCurrentTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.setCurrentTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.getCurrentTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpServletRequest.getInputStream"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#startYear">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.numberOfDays"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startYear"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startDayOfYear"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.beginDate"/>
```

```
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#startDayOfYear">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.numberOfDays"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startYear"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startDayOfYear"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.beginDate"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflow"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeServlet"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Project"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#contains">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.containsDate"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.containsUser"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.containsClient"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#clientId">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.client"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.clients"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper.clientName"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#description">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Client"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#IConcreteProducer.TABULAR_INPUT_FORM_PRODUCER"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.description"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.description"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.description"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TabularInputFormProducer.formProducer"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.description"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#IConcreteProducer.FORM_PRODUCER"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Node.description"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Activity"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextEntryFrame"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Client">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Node"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Client"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
```

```
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#hours">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.hours"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.hoursString"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddEmployeeWorkflowHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddEmployeeWorkflowBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddEmployeeWorkflow"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#has">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.buildHash"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#userId">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.newUser"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.employee"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.users"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#id">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Client"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#numberOfDays">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.numberOfDays"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startYear"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.startDayOfYear"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#password">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.password"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#exportRequest">
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.exportFile"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#currentTimecardId">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecard"/>
```

```
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.projectId"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.employeeId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ClientBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.timecard"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.clientId"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodeIds"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean.id"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProjectBean.id"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.currentTimecardId"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#represents">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Activity"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#submitTimecard">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.setCurrentTimecard"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#TimeRecord"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflow.submitTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#SubmitButtonProducer.setSubmittLabel"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#FormProducer.setSubmitTarget"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.ExportTimeEntriesApplication"
/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.exportEntries"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Employee.getCurrentTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ProducerFactory.getSubmitButtonProducer"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportFile.addEntry"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardHome.findTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#FormProducer.getSubmitTarget"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TabularInputFormProducer.setSubmitLabel"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TabularInputFormProducer.setSubmitTarget"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.setCurrentTimecard"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#User"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TextEntryFrame.TextEntryFrame"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.addTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#EmployeeBean.getCurrentTimecard"/>
    <j.0:ehAssociadaA
rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.submitTimecard"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.TimecardBean"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Timecard">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#Timecard"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#associatedTo">
    <rdfs:range rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#Project"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/2012/4/Timecard.owl#ChargeCode">
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeWrapper"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflow"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeHome"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeBean"/>
    <j.0:ehAssociadaA rdf:resource="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCode"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
</rdf:RDF>
```

Ontologia Parcial do Código Fonte

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityBean.getName">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#Activity.getDescription">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#Timecard.addChargeCode">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflow.isUserValid">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.chargeCodes">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TextFieldProducer">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpSession.getValue">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TimecardBean.loadChargeCodes">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TestHttpSession.putValue">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TabularInputFormProducer.customizePreferences">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#RecordTimeWorkflowBean.timecard">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ComboBoxProducer.addValue">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#FormProducer.addHtmlProducer">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#AddChargeCodeWorkflowBean.addProject">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#FormProducer.method">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#LoginWorkflow.setPassword">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ProducerFactory.factory">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ExportTimeEntriesApplication.addTimecard">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ExportCriteria.getUsers">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#Client.getDescription">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ActivityHome.create">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#ChargeCodeHome.findByPrimaryKey">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#TextFieldProducer.getMaxLength">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://iseg.pucrs.br/SEmantics/SourceCode#Timecard.getHours">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>

   …

</rdf:RDF>
```

APÊNDICE E. INSTRUMENTAÇÃO DO PRIMEIRO EXPERIMENTO

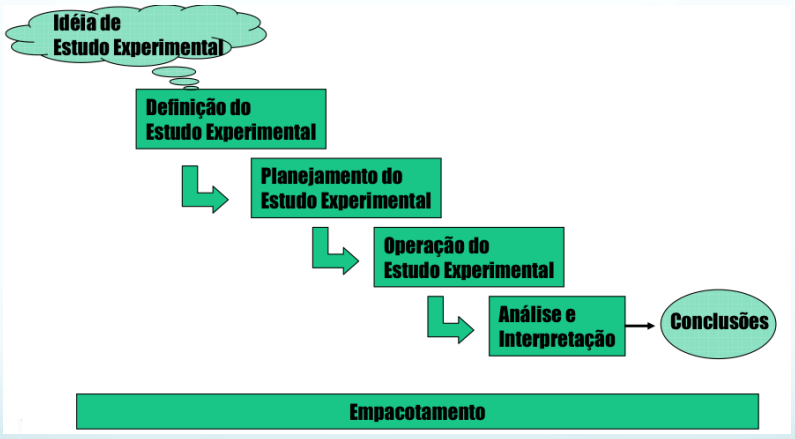
PUCRS

Um modelo para análise de impacto em código fonte utilizando ontologias

Engenharia de Software Experimental

Rodrigo Perozzo Noll
Orientador: D.Sc. Marcelo Blois Ribeiro

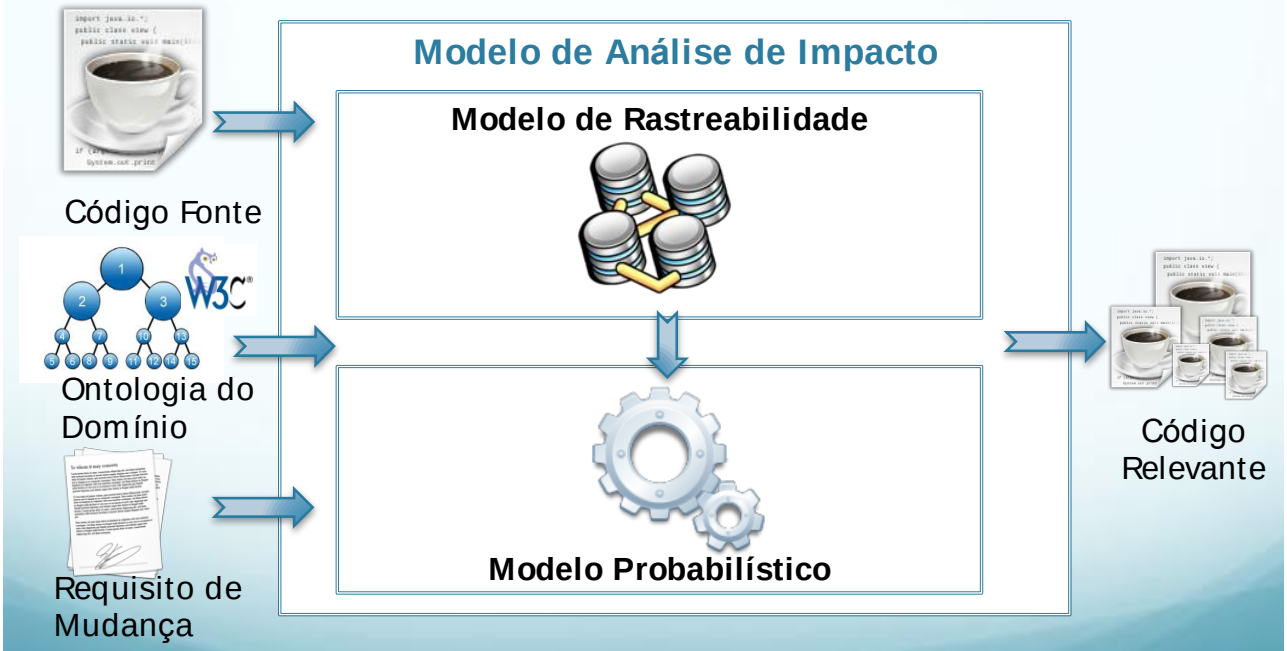
Engenharia de Software Experimental



Ideia de Estudo Experimental

- **Análise de Impacto**
  - A análise de impacto corresponde a identificação de **potenciais consequências** de uma mudança ou a estimativa do que precisa ser **modificado** com o objetivo de realizar uma mudança [ARN96]
- **Métodos para a análise de impacto [ARN93]:**
  - a navegação por um programa
  - pesquisa em especificações
  - listas de referências cruzadas
  - modelo de rastreabilidade
  - modelo de dependência

Ideia de Estudo Experimental



Definição

- **Objetivo Global**
  - Comparar a precisão da análise de impacto de estruturas de código fonte alteradas por uma requisição de mudança considerando uma abordagem manual e automatizada.
- **Questão de Pesquisa:**
  - A precisão da análise de determinada mudança no código fonte de uma aplicação é a mesma através da inspeção manual e automatizada?

Definição

- *Comparar* a análise de impacto manual com a análise de impacto automatizada obtida pela ferramenta SEmantics,
- *Com o propósito de* avaliar a eficiência de ambas as abordagens,
- *Com foco* na precisão e revocação,
- *Sob o ponto* de vista de um programador,
- *No contexto* de uma implementação de um requisito de mudança durante a manutenção evolutiva de software.

Definição

- **Objetivo de Medição**
  - Definir a precisão e revocação considerando a completude e corretude da identificação das estruturas de código através da inspeção manual e automatizada de quais classe serão impactados por uma requisição de mudança.

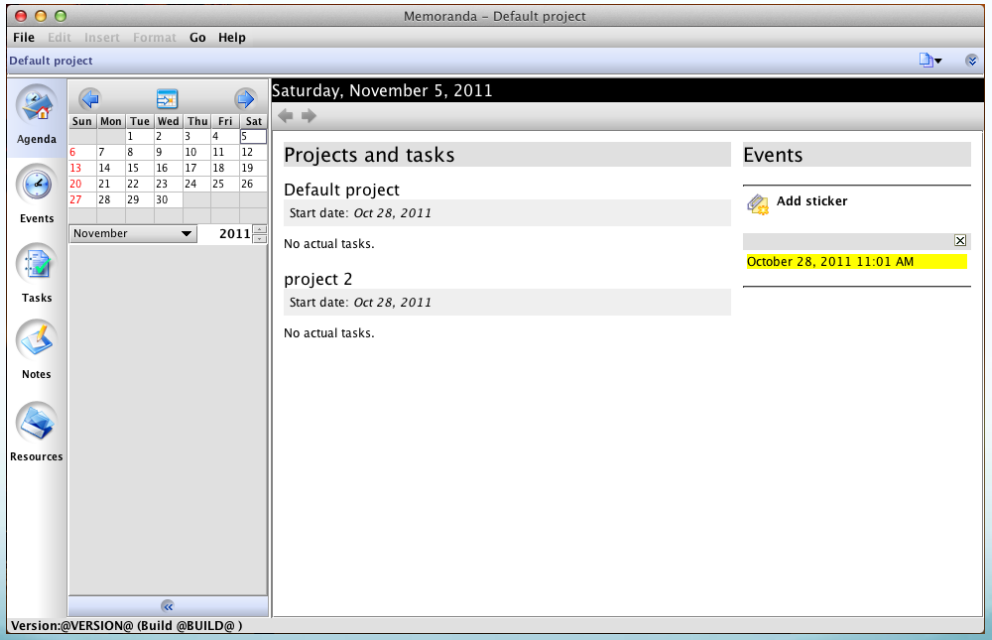
Planejamento

- **Objetos fornecidos:**
  - Código fonte da aplicação executando em uma IDE
  - Solicitação de mudança
  - Formulário para definição dos métodos e/ ou atributos impactados por determinada mudança.
- **Guias**:
  - Treinamento
- **Coleta das métricas**:
  - Através de formulários preenchido pelos participantes.

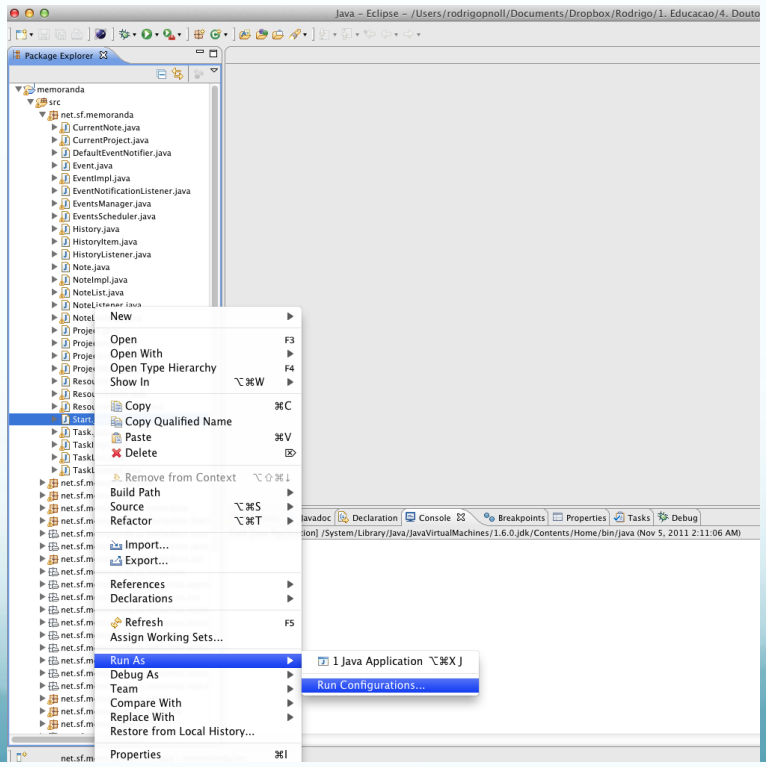
Execução

- **Memoranda** (*http://memoranda.sourceforge.net*)
  - Trata-se de uma ferramenta de organização pessoal, multiplataforma, desenvolvida em Java e com armazenamento local em XML.
  - Tem como objetivo gerenciar:
    - **Projetos**: representa uma atividade que precisa ser organizada, com data de início e fim. Pod-se criar, excluir e modificar projetos. Ex.: se preparar para uma prova
    - **Tarefas**: atividade semelhante a lista de "to-do" e pode ser subdividida em outras tarefas, bem como acompanhar seu andamento.
    - **Eventos**: são como lembretes de compromissos que podem ser agendados com regularidade (repetição)
    - **Notas**: espaço reservado para anotações simples
    - **Recursos**: pemite adicionar recursos locais (documentos) no diretório do software ou referências a URLs.

Execução

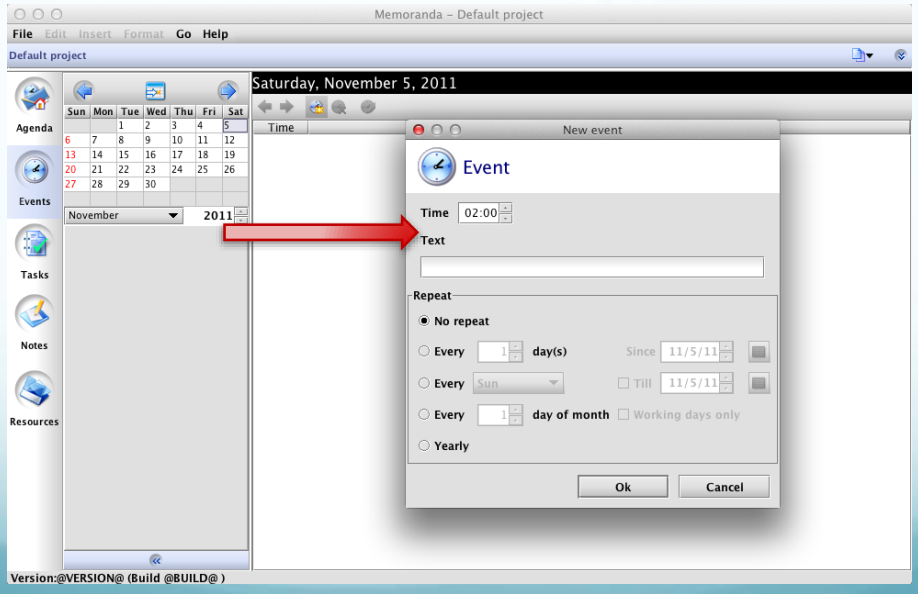


Execução



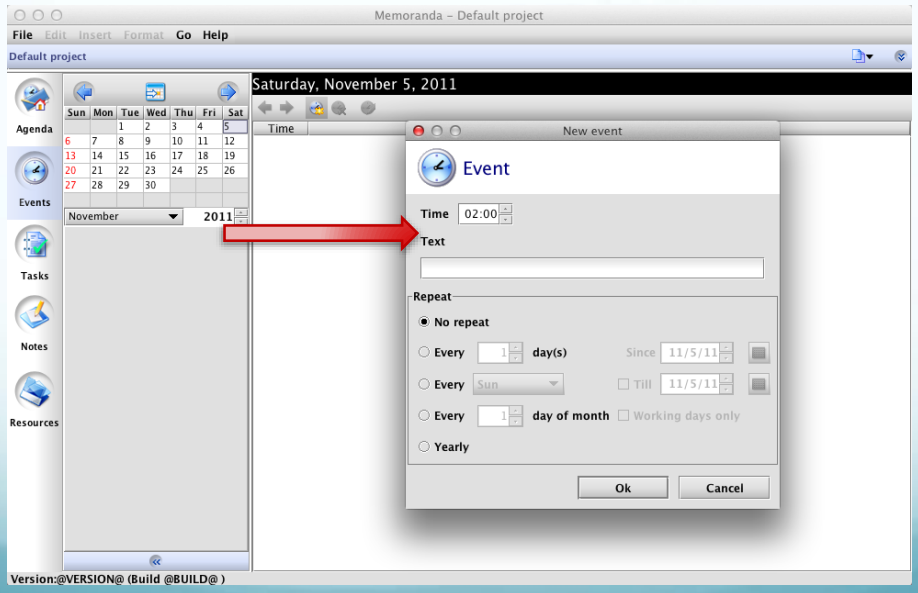
Requisito de Mudança #1

- Permitir ao usuário **informar a data de um evento**.



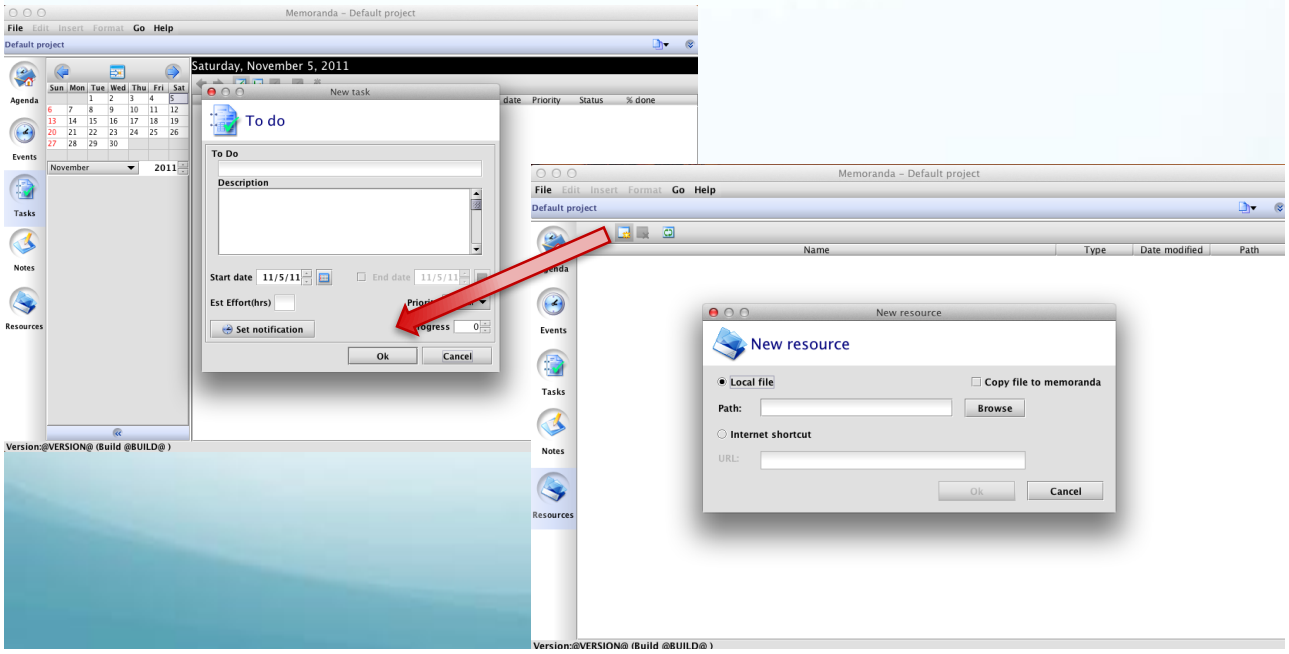
Requisito de Mudança #2

- Permitir ao usuário **informar uma descrição**.



Requisito de Mudança #3

- Adicionar **Recursos a uma Tarefa**



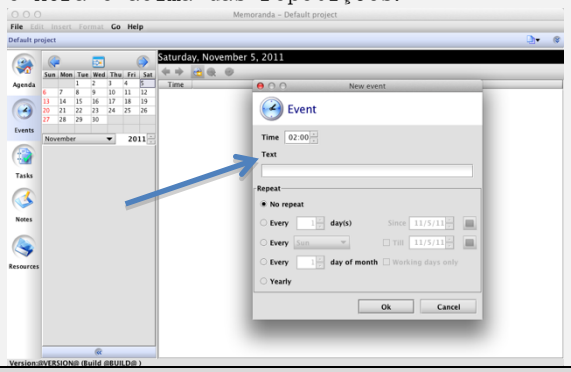
Execução

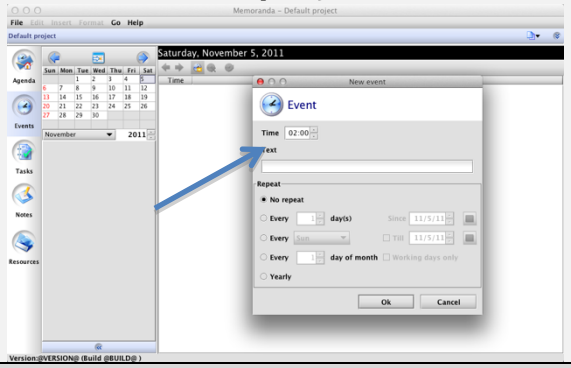
- **Instruções Gerais:**
  - Deve-se fazer o **registro do início e término de cada análise**;
  - Trabalho em duplas;
  - Não é possível a interação entre as duplas;
  - Manter os celulares desligados;
  - Evitar a interação com ambiente externo ao experimento;
  - Quaisquer dúvidas sobre os procedimentos, perguntar exclusivamente para a equipe responsável;

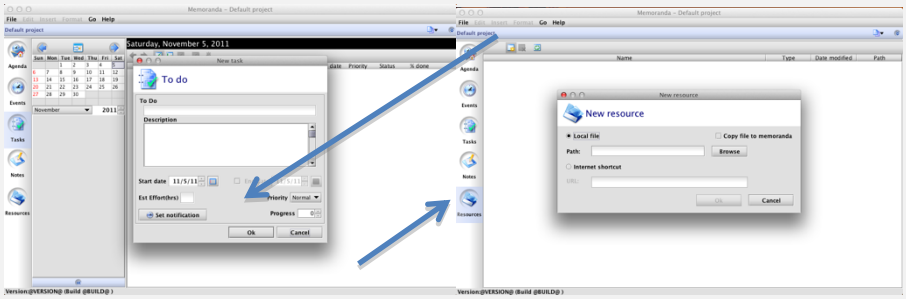
Execução

- **Observações:**
  - O resultado esperado não demanda rapidez, mas atenção e qualidade;
  - Durante a documentação do experimento, os participantes serão referenciados de forma anônima. Não haverá nenhuma referência a nomes.
- Não seremos capazes de apresentar o resultado da experimentação hoje.
- É necessário a análise estatística dos dados coletados pelas duas abordagens (manual e automatizada)
- Para os interessados, posteriormente podemos apresentar os resultados obtidos;

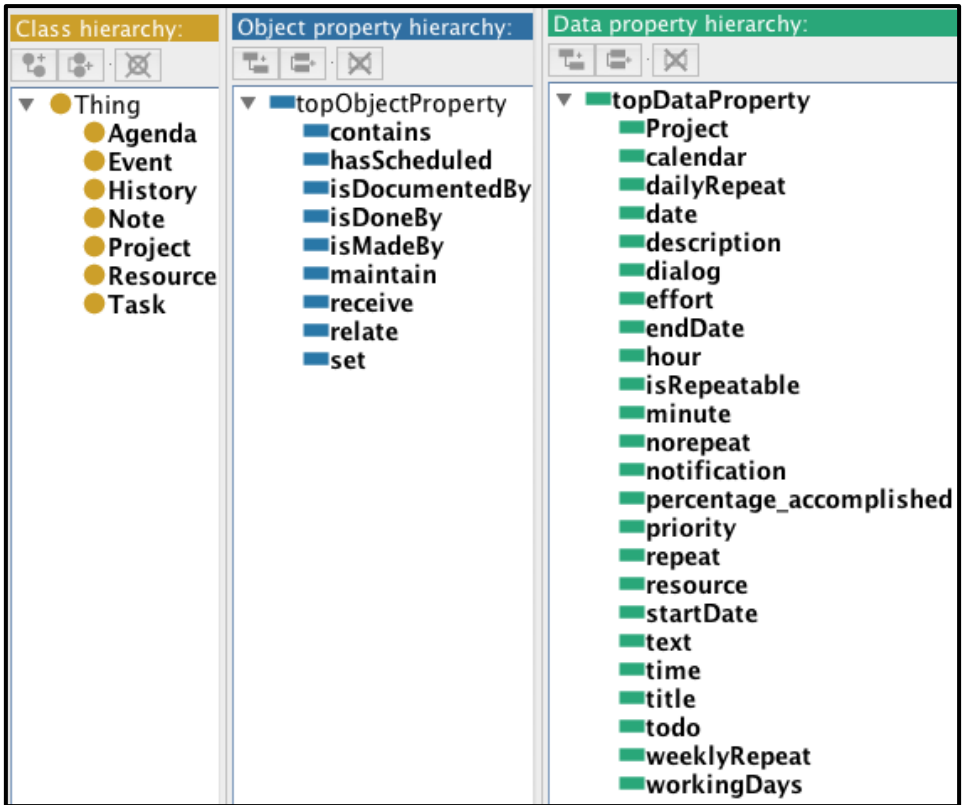
Relatório de Análise de Impacto

Nomes	
Hora de Início	
Hora de Término	
Análise de Impacto no Projeto	
Descrição da Alteração	Permitir ao usuário informar a data de um evento logo abaixo do texto e hora e acima das repetições. 
Código Fonte Impactado	Descrição do Impacto
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
Observações	

Hora de Início	
Hora de Término	
Análise de Impacto no Projeto	
Descrição da Alteração	O usuário pode adicionar uma descrição do evento abaixo do texto e hora e acima das repetições. 
Código Fonte Impactado	Descrição do Impacto
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
Observações	

Hora de Início	
Hora de Término	
Análise de Impacto no Projeto	
Descrição da Alteração	Adicionar recursos a janela de tarefas próximo ao TODO, descrição, esforço, datas, notificação, prioridade e progresso. 
Código Fonte Impactado	Descrição do Impacto
<Nome da classe> - <alterado/incluído/excluído>	<descrição do impacto no código fonte alterado/incluído/excluído.>
Observações	

Ontologia


Class hierarchy:
Thing
Agenda
Event
History
Note
Project
Resource
Task
Object property hierarchy:
topObjectProperty
contains
hasScheduled
isDocumentedBy
isDoneBy
isMadeBy
maintain
receive
relate
set
Data property hierarchy:
topDataProperty
Project
calendar
dailyRepeat
date
description
dialog
effort
endDate
hour
isRepeatable
minute
norepeat
notification
percentage_accomplished
priority
repeat
resource
startDate
text
time
title
todo
weeklyRepeat
workingDays

APÊNDICE F. INSTRUMENTAÇÃO DO SEGUNDO EXPERIMENTO

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação
Grupo de Engenharia de Sistemas Inteligentes

Engenharia de Software Experimental

Mapeamento entre especificação de software e modelagem conceitual

ISEG

Rodrigo Noll

Agenda

- Definição
- Planejamento
- Execução
- Observações Gerais

Definição

- **Comparar** o mapeamento manual obtido por seres humanos não especialistas no domínio da aplicação, com o mapeamento automático obtido pela ferramenta ONTraceMatching,
- **Com o propósito de** avaliar a eficácia de ambas as abordagens,
- **Com foco** na precisão,
- **Sob o ponto de vista** de um especialista do domínio,
- **No contexto** da definição de um mapeamento entre especificações de software e conceitos do domínio.

Planejamento - Instrumentação

- Objetos fornecidos:
  - Especificação de Requisitos;
  - Formulário para mapear a especificação de requisitos com os conceitos do domínio.
- Guias:
  - Treinamento.
- Coleta das métricas:
  - Através de formulários preenchido pelos participantes.

Execução: Modo de Operação

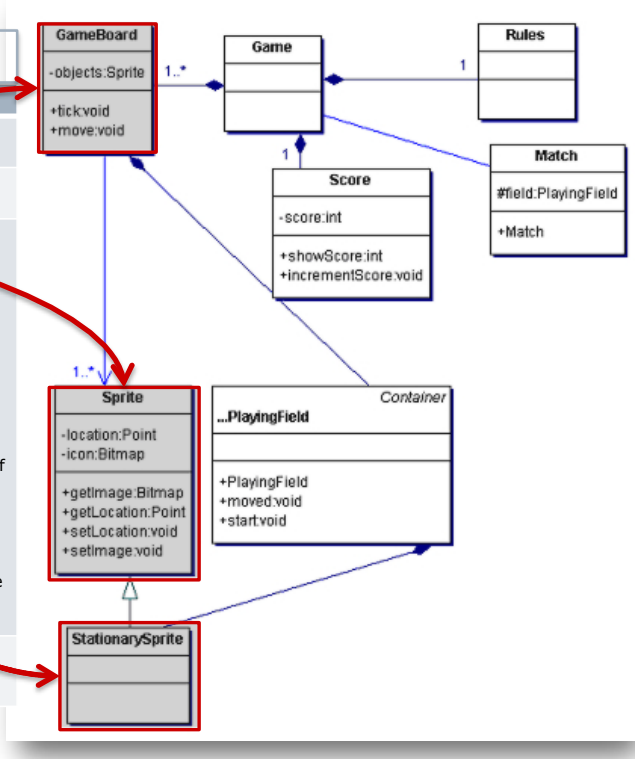
Percorrer a especificação de requisito com o objetivo de identificar quais conceitos do domínio estão presentes nesta especificação.

Execução: Modo de Operação

Pong-like arcade game, a.k.a. Arkanoid/Brickles

Execução: Definição do Mapeamento

Actor	System Response
Selects PLAY from the menu	Initializes the game and displays the gameboard
Left-clicks to begin play	Starts game action
Left-clicks or uses the keyboard to enter commands	Moves the paddle horizontally to follow the mouse track After each movement of the puck, system checks for a collision with another object. If puck collides with the ceiling, it is reflected back into the playing area. If the puck collides with a wall, it is reflected back into the playing area. If the puck collides with the floor, it ceases to exist. If the maximum number of pucks has not been reached, requests and provides a new puck. If the maximum has been reached, the Lost dialog is presented. If the puck collides with a brick, defines action by the type of brick. When the puck collides with the last brick, the Won dialog is presented.
Responds to Won/ Lost dialog box with left-click	Returns the gameboard to its initialized, ready-to-play state



Fonte: http://www.sei.cmu.edu/productlines/ppl/requirements_model.html

Execução: Definição do Mapeamento

ARTIFACTS	Game Board	Game	Rules	Sprite	Stationary Sprite
UC01 - Play Brickles	⊗	⊗		⊗	⊗

Fonte: http://www.sei.cmu.edu/productlines/ppl/requirements_model.html

Execução: Definição do Mapeamento

Não esquecer de preencher a hora de início e término do processo de mapeamento dos elos.

Hora de Início:	
Hora de Término:	

Observações Gerais

- Instruções Gerais:
  - Deve-se fazer o registro do início e término da atividade;
  - Não é possível a interação entre os grupos;
  - Manter os celulares, IM, etc, desligados;
  - Evitar a interação com ambiente externo ao experimento;
  - Quaisquer dúvidas sobre os procedimentos, perguntar exclusivamente para a equipe responsável;

Observações Gerais

- O resultado esperado não demanda rapidez, mas atenção e qualidade;
- Durante a documentação do experimento, os participantes serão referenciados de forma anônima. Não haverá nenhuma referência a nomes.

Observações Gerais

- Não seremos capazes de apresentar o resultado da experimentação hoje.
- É necessário a análise estatística dos dados coletados pelas duas abordagens de indexação (requisitos e conceitos)
- Para os interessados, posteriormente podemos apresentar os resultados obtidos;

Casos de Uso

UC01. Create Charge Code

Description. The administrative user actor uses the Create Charge Code use case to populate the time-tracking system with charge codes. Once added, a charge code is available to all employees as they enter their hours.

Preconditions. None

Deployment constraints. None

Normal flow of events. *Add a charge code to an existing project*.

1. The administrative user sees a view of existing charge codes. Charge codes are activities organized by client and project.
2. The administrative user adds a charge code to an existing project. The new charge code appears in the view, and may be used by employees.

Alternate flow of events. *New charge code for a new project for a new client.*

1. The administrative user sees a view of existing charge codes. Charge codes are activities organized by client and project.
2. The administrative user adds a client. The new client appears in the view.
3. The administrative user adds a project to the new client. The new project appears in the view.
4. The administrative user adds a charge code to the new project. The new charge code appears in the view and may be used by employees.

Alternate flow of events. *Duplicate charge code.*

1. The administrative user sees a view of existing charge codes. Charge codes are activities organized by client and project.
2. The administrative user adds a charge code to an existing project. The charge code already exists for the project.
3. The system informs the administrative user that the charge code already exists. No change to the view.

Exception flow of events. *System is unable to add the charge code due to a system or communication error.*

1. The administrative user sees a view of existing charge codes. Charge codes are activities organized by client and project.
2. The administrative user adds a charge code to an existing project. The system is unable to complete the addition, due to a system or communication error.
3. The system informs the administrative user the error, complete with available details. The view reverts to the previous state.
4. If possible, an error is added to a log.

Nonfunctional requirements. None

UC02 Create Employee

Description. The Create Employee use case allows the user to add an employee to the time-tracking system. Once employees have been created, they are able to use the system to record their time.

Preconditions. None

Deployment constraints. None.

Normal flow of events. *The administrator adds an employee*.

1. The administrator sees a view of all existing employees by name.
2. The administrator adds an employee, with a name, email address, and password.
3. The new employee appears in the view. The employee can record his or her hours.

Alternate flow of events. *Employee exists.*

1. The administrator sees a view of all existing employees by name.
2. The administrator adds an employee, with a name, email address, and password.
3. The administrator is notified of the conflict. No change to existing data.

Exception flow of events. *System is unable to add the employee due to a system or communication error.*

1. The administrator sees a view of all existing employees by name.
2. The administrator adds an employee, with a name, email address, and password. The system is unable to complete the addition, due to a system or communication error.
3. The system informs the administrator of the error, complete with available details. The view reverts to the previous state.
4. If possible, an error is added to a log.

Nonfunctional requirements. None

UC03 Record Time

Description. The Record Time use case allows employees to track the hours that they work. The Record Time use case allows an administrator to record hours for any employee.

Preconditions. None

Deployment constraints. The Record Time use case must be accessible from client sites and the employees' homes. In the case of client sites, they will often be behind the client's firewall.

Normal flow of events. *An employee records his or her time.*

1. The employee sees previously entered data for the current time period.
2. The employee selects a charge number from all available charge numbers.
3. The employee selects a day from the current week.
4. The employee enters the hours worked as a positive decimal number.

5. The new hours are added to the view and are seen in any subsequent views.

Alternate flow of events. *An employee updates his or her time.*

1. The employee sees previously entered data for the current week.
2. The employee selects an existing entry.
3. The employee changes the hours worked.
4. The new information is updated in the view and is seen in any subsequent views.

Alternate flow of events. *An administrator records time for an employee*.

1. The administrator is presented with a list of employees, sorted by name.
2. The administrator selects an employee and sees previously entered data for the current time period.
3. The administrator selects a charge number from all available charge numbers.
4. The administrative user selects a day from the current time period.
5. The administrator enters the hours worked as a positive decimal number.
6. The new hours are added to the view and are seen in any subsequent views.

Exception flow of events. *System is unable to add the update to the timecard due to a system or communication error.*

1. The employee sees previously entered data for the current time period.
2. The employee selects a charge number from all available charge numbers, organized by client and project.
3. The employee selects a day from the current time period.
4. The employee enters the hours worked as a positive decimal number. The system is unable to complete the addition, due to a system or communication error.
5. The system informs the administrative user of the error, complete with available details. All additions and edits are undone together. The view reverts to the previous state.
6. If possible, an error is added to a log.

Nonfunctional requirements. None

UC04 Export Time Entries

Description. The Export Time Entries use case allows the administrative user to save specified time-tracking data to a formatted file.

Preconditions. None

Normal flow of events. *The administrator exports the data.*

1. The administrator selects a range of dates.
2. The administrator selects a subset of clients or all.
3. The administrator selects a subset of employees or all.
4. The administrator selects a target file.
5. The data is exported to the file as XML. The administrator is notified when the process is complete.

Exception flow of events. *System is unable to export the data due to a system error.*

1. The administrator selects a range of dates.
2. The administrator selects a subset of clients or all.

3. The administrator selects a subset of employees or all.
4. The administrator selects a target file.
5. The system is unable to export the data. The administrative user is notified of the error
6. If possible, the error is recorded to a log.

Nonfunctional requirements. None

UC05 Login

Description. The Login use case allows users to access the system.

Preconditions. None

Deployment constraints.

1. Employees must be able to log in from any computer, including home, client sites, and on the road. This access may be from behind a client's firewall.

Normal flow of events. *The user's username and password are valid.*

1. The user supplies a username and password.
2. The user is authenticated as either an administrator or an employee. This is not a choice during the login; it is determined by the username.

Alternate flow of events. *First Login*

1. The user supplies a username and password.
2. The user is authenticated as either an administrator or an employee. This is not a choice during the login; it is determined by the username.
3. The user is instructed to change his or her password.
4. Include the Change Password use case at this point.

Alternate flow of events. *Invalid authentication information*.

1. The user supplies a username and password.
2. The user is notified that he or she has entered incorrect login information.
3. The failure is logged by the system.
4. The user is allowed to try again indefinitely.
5. The users password must not be passed as plaintext.

UC06 Change Password

Description. The Change Password use case allows users to change their password.

Preconditions. The user must have logged in to the system.

Deployment constraints. Employees and administrative users must be able to log in from any computer, including home, client sites, and on the road. This access may be from behind a client's firewall.

Normal flow of events. *Employee changes his or her password*.

1. The user enters his or her current password and new password twice.

2. The user is notified that his or her password has been changed.

Alternate flow of events. *Invalid current password*.

1. The user enters his or her current password and new password twice.
2. The user is notified that his or her attempt failed.
3. The failure is logged by the system.
4. The user is allowed to try again indefinitely.

Alternate flow of events. *New passwords do not match*.

1. The user enters his or her current password and new password twice,
2. The user is notified that his or her attempt failed.
3. The user is allowed to try again indefinitely.

Exception flow of events. *System is unable to store new password due to a system or communications error.*

1. The user enters his or her current password and new password twice.
2. The user is notified of the error, complete with any available details.
3. The failure is logged by the system.
4. The user's password must not be passed as plaintext

Matriz de Rastreabilidade

Informações Pessoais

1. Nome		
2. Escolaridade (maior grau)	☐ Superior ☐ Especialização ☐ MBA ☐ Mestrado ☐ Doutorado	☐ Incompleto ☐ Completo
3. Experiência profissional na área de engenharia de software		(anos)
4. Hora de início:		
5. Hora de Termino:		

Atividade

Com base nas seis especificações de casos de uso, relacione os conceitos do domínio associados:

Conceito	UC1	UC2	UC3	UC4	UC5	UC6
User						
Timecard						
TimeRecord						
Activity						
Client						
ChargeCode						
Project						

APÊNDICE G. RESULTADOS DO SEGUNDO EXPERIMENTO

CdU	Particip.	Precisão	Revocação	Medida F	CdU	Particip.	Precisao	Revocacao	Medida F
1	P1	1,00	0,83	0,91	4	P1	1,00	0,50	0,67
	P2	1,00	0,67	0,80		P2	1,00	0,25	0,40
	P3	1,00	0,83	0,91		P3	1,00	0,50	0,67
	P4	1,00	0,83	0,91		P4	1,00	0,25	0,40
	P5	1,00	0,83	0,91		P5	1,00	0,50	0,67
	P6	1,00	0,83	0,91		P6	1,00	0,50	0,67
	P7	1,00	1,00	1,00		P7	1,00	0,50	0,67
	P8	1,00	0,83	0,91		P8	1,00	0,50	0,67
	P9	1,00	0,50	0,67		P9	1,00	0,25	0,40
	P10	1,00	0,83	0,91		P10	1,00	0,38	0,55
	P11	1,00	0,83	0,91		P11	1,00	0,50	0,67
	P12	1,00	0,67	0,80		P12	1,00	0,75	0,86
	P13	1,00	0,83	0,91		P13	1,00	0,50	0,67
	M(µman)	**1,00**	**0,79**	**0,88**		**M(µman)**	**1,00**	**0,45**	**0,61**
	SEmantics	1,00	1,00	1,00		SEmantics	1,00	0,75	0,86
	M(µaut)	**1,00**	**1,00**	**1,00**		**M(µaut)**	**1,00**	**0,75**	**0,86**
2	P1	1,00	0,67	0,80	5	P1	0,50	1,00	0,67
	P2	1,00	0,67	0,80		P2	0,00	0,00	0,00
	P3	1,00	1,00	1,00		P3	0,67	1,00	0,80
	P4	0,75	1,00	0,86		P4	0,67	1,00	0,80
	P5	1,00	1,00	1,00		P5	1,00	1,00	1,00
	P6	1,00	1,00	1,00		P6	1,00	1,00	1,00
	P7	1,00	0,67	0,80		P7	0,67	1,00	0,80
	P8	0,75	1,00	0,86		P8	0,33	0,50	0,40
	P9	1,00	0,33	0,50		P9	1,00	1,00	1,00
	P10	1,00	0,67	0,80		P10	1,00	1,00	1,00
	P11	1,00	1,00	1,00		P11	1,00	1,00	1,00
	P12	1,00	0,67	0,80		P12	1,00	1,00	1,00
	P13	1,00	0,67	0,80		P13	1,00	1,00	1,00
	M(µman)	**0,96**	**0,79**	**0,85**		**M(µman)**	**0,76**	**0,88**	**0,81**
	SEmantics	1,00	1,00	1,00		SEmantics	0,67	1,00	0,80
	M(µaut)	**1,00**	**1,00**	**1,00**		**M(µaut)**	**0,67**	**1,00**	**0,80**
3	P1	1,00	0,63	0,77	6	P1	0,67	1,00	0,80
	P2	1,00	0,63	0,77		P2	0,00	0,00	0,00
	P3	1,00	0,88	0,93		P3	0,67	1,00	0,80
	P4	1,00	0,75	0,86		P4	0,67	1,00	0,80
	P5	1,00	0,63	0,77		P5	1,00	1,00	1,00
	P6	1,00	0,88	0,93		P6	1,00	1,00	1,00
	P7	1,00	0,75	0,86		P7	0,67	1,00	0,80
	P8	1,00	0,75	0,86		P8	0,50	1,00	0,67
	P9	1,00	0,50	0,67		P9	1,00	1,00	1,00
	P10	1,00	0,75	0,86		P10	0,67	1,00	0,80
	P11	1,00	0,63	0,77		P11	1,00	1,00	1,00
	P12	1,00	0,88	0,93		P12	1,00	1,00	1,00
	P13	1,00	0,75	0,86		P13	1,00	0,50	0,67
	M(µman)	**1,00**	**0,72**	**0,83**		**M(µman)**	**0,76**	**0,88**	**0,79**
	SEmantics	1,00	1,00	1,00		SEmantics	1,00	1,00	1,00
	M(µaut)	**1,00**	**1,00**	**1,00**		**M(µaut)**	**1,00**	**1,00**	**1,00**

APÊNDICE H. INSTRUMENTAÇÃO DO TERCEIRO EXPERIMENTO

Um modelo para análise de impacto em código fonte utilizando ontologias

Engenharia de Software Experimental

Rodrigo Perozzo Noll

Definição

- *Comparar o tempo despendido com a implementação de duas mudanças,*
- *Com o propósito de avaliar o desempenho da ferramenta SEmantics,*
- *Com foco no esforço,*
- *Sob o ponto de vista de um programador de software não especialista no domínio da aplicação,*
- *No contexto de uma manutenção corretiva.*

Planejamento

- **Objetivo de Medição**
  - Definir o tempo total gasto para a implementação de mudanças nas estruturas de código através da inspeção manual e automatizada durante a evolução do software.
- **Objetos fornecidos:**
  - Código fonte da aplicação executando em uma IDE
  - Solicitação de mudança
  - Formulário para registro do tempo de início e término das atividades.
- **Guias**:
  - Treinamento
- **Coleta das métricas**:
  - Através de formulários preenchido pelos participantes.

Execução

- **Modo de Operação**
  - Registrar início da atividade.
  - Compreender o requisito de mudança e implementar as mudanças.
  - Registrar o término da atividade.

Execução

- **Locadora Online**
  http://javafree.uol.com.br/artigo/869464/Ex-praticoProject-in-NetBeans-Conceito-MVC+Servlet+JSP.html

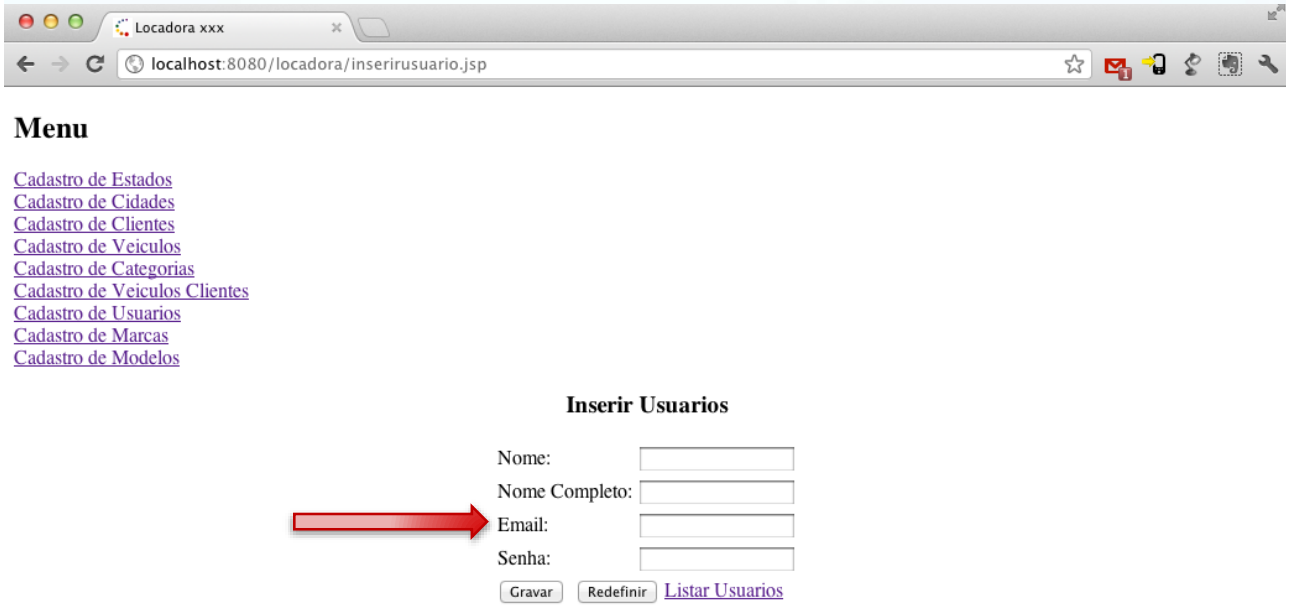
  - Trata-se de uma ferramenta online para gerenciar locação de veículos, incluindo:
    - Cadastro de Estados
    - Cadastro de Cidades
    - Cadastro de Clientes
    - Cadastro de Veiculos
    - Cadastro de Categorias
    - Cadastro de Veiculos Clientes
    - Cadastro de Usuarios
    - Cadastro de Marcas
    - Cadastro de Modelos

Requisito de Mudança #1

- Quando cadastrar ou alterar o modelo, garantir que a descrição não tenha mais que 10 caracteres

Requisito de Mudança #2

- Quando inserir usuário ou alterar usuário, validar se a senha possui mais que 6 caracteres



Execução

- **Instruções Gerais:**
  - Deve-se fazer o **registro do início e término de cada análise**;
  - Trabalho em duplas;
  - Não é possível a interação entre as duplas;
  - Manter os celulares desligados;
  - Evitar a interação com ambiente externo ao experimento;
  - Quaisquer dúvidas sobre os procedimentos, perguntar exclusivamente o responsável;

ANEXO A. TABELA DE DISTRIBUIÇÃO DE T

Graus de Liberdade	Probabilidade, p a 0.05
1	12.71
2	4.30
3	3.18
4	2.78
5	2.57
6	2.45
7	2.37
8	2.31
9	2.26
10	2.23
11	2.20
12	2.18
13	2.16
14	2.14
15	2.13
16	2.12
17	2.11
18	2.10
19	2.09
20	2.09
21	2.08
22	2.07
23	2.07
24	2.06
25	2.06
26	2.06
27	2.05
28	2.05
29	2.05
30	2.04
40	2.02
60	2.00
120	1.98
∞	1.96